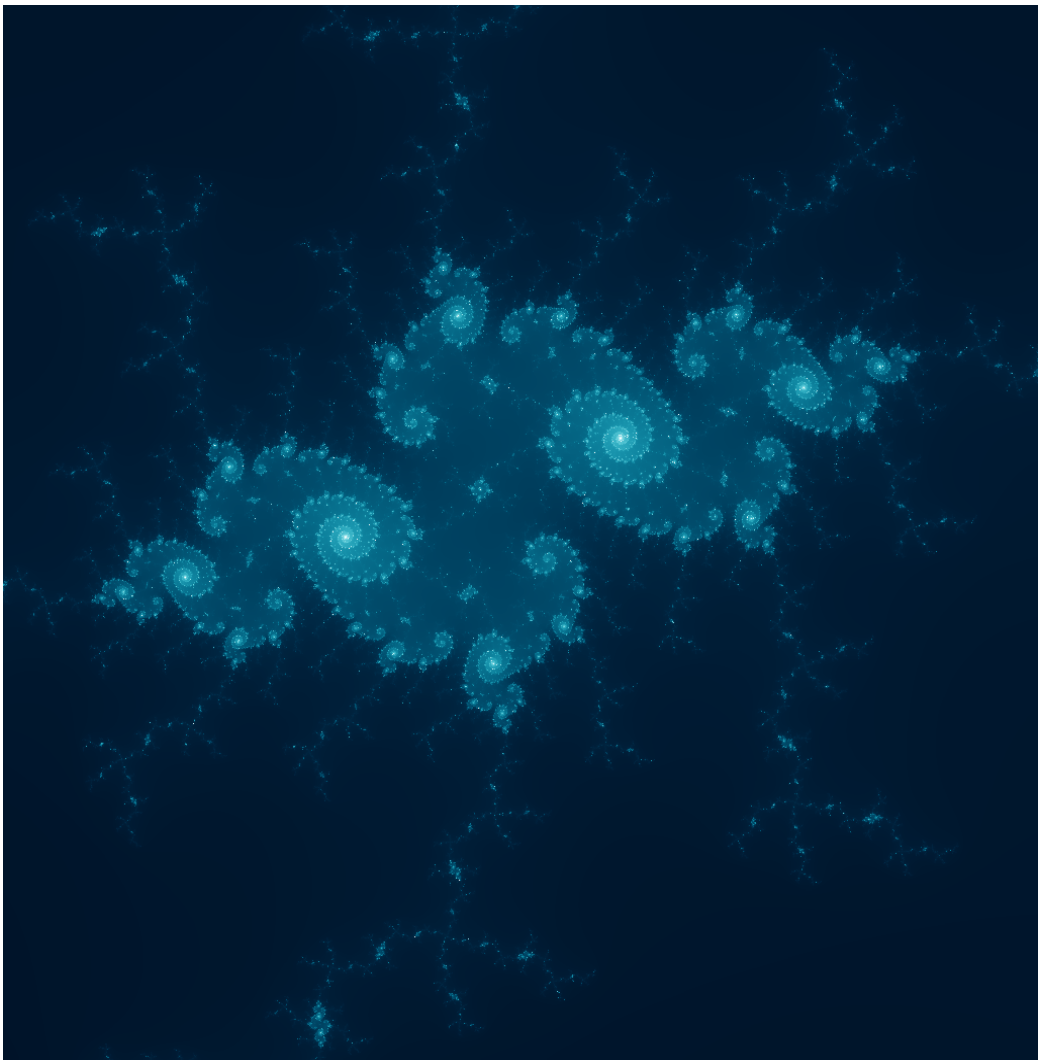
easel



Manager | Danielle Crosswell | dac2182
Language Guru | Tyrus Cukavac | thc2125
System Architect | Yuan-Chao (Oswin) Chou | yc3211
Tester | Xiaofei (Sophie) Chen | xc2364

1. Introduction

We created a language that allows a user to create art using mathematics. By writing a few simple functions, a user is able to combine basic mathematical principles with colors to create aesthetically pleasing visualizations. easel takes advantage of the fact that images are essentially matrices of pixels, and thus the user can easily create geometrical and symmetric images by simply manipulating certain pixel values.

Because creating visualizations is the main goal of easel, in addition to basic data types including integers, floats, and booleans, our language includes a pix datatype that is able to represent pixel values in integer form. Additionally, our language can implement a two-dimensional array, which allows the user to create the canvas on which to draw the image.

The goal of easel is to create a simple interface for mathematical data and functions to be transformed into images, and so easel is capable of performing both basic and advanced arithmetic, such as trigonometric and logarithmic functions using built-in functions and simple operators. Additionally, functions are the core of the language and so they can be passed as parameters and declared anonymously. Our hope is to be able to make complex images while keeping the programming as simple as possible.

2. Language Tutorial

2.1 Installing Dependencies and Running the Compiler

The first step in running easel programs is downloading the proper dependencies. In order output easel drawings, you must have OpenGL installed on your machine. To install all necessary libraries, run the following bash command:

```
$ sudo apt-get install freeglut3 freeglut3-dev libglew-dev gle-graphics
```

Now use the provided source files or download from the GitHub repository:

```
$ git clone https://github.com/OswinC/easel.git
```

Run make in the glwrap/ directory and the easel/ directory

```
$ cd glwrap
$ make
$ cd ../
$ make
```

To compile and run a program:

```
$ ./ecc.sh filename.es
$ ./filename
```

2.2 Basic Program

The following is a program called squares.es that will create a checkered pattern with user-defined size squares and colors. The program will take a matrix and fill in the corresponding pixels with the given colors, based on the size argument passed in by the user. By calling the draw built-in function, the checkered patter will be drawn onto the user's screen.

Start by creating a file called squares.es and opening it up in a text editor.

First, define global variables at the top of the program. In this program, we need two colors to alternate in squares (so two pix data types), a matrix that can hold pixel values and integer values that store the width and height of the matrix that we defined. It is important to note that we are defining pix values in two different ways in this example (and both are valid). Variables are also able to be declared on the same line, separated by a comma.

```
pix col1 = #BF8FD600;
pix col2 = {70, 58, 39, 170};
pix canvas[960][720];
int W = 960, H = 720;
```

Next we are going to define a function (we will first look at the function definition and then look at the body of the function). The function below is named plaid, takes in one integer argument, and returns void.

```
function void plaid(int p_w) {
    /* function body */
}
```

Looking at the function body, we first declare our local variables and then create nested for loops to iterate through all values in the pix matrix. Without looking at the body of the nest for loops, the syntax is as follows. For loops take three expressions and increment the given variable until it meets the condition given in the second expression of the loop.

```
int x, y;
for (y = 0; y < H; y++) {
    for (x = 0; x < W; x++) {
        /* body */
    }
}
```

Next, the inside of the for loop consists of an if-else statement. This statement will check if the given condition is true (this is where the program determines which color to assign to the given index of the matrix). Because the statements inside of the if-else statements is only one line, braces are not necessary (and are omitted for cleaner code).

```
if (((x/p_w)%2) == ((y/p_w)%2))
    canvas[x][y] = col1;
else
    canvas[x][y] = col2;
```

Finally, we still need to actually call the plaid function in the program and draw the canvas returned from the function. We do this simply by making two function calls: the first to plaid (where the argument is an integer that specifies how large to make the returned squares) and the second to draw function, which takes as arguments a matrix and x and y values that specify offset from the top left corner.

```
plaid(100);
draw(canvas, 0, 0);
```

Finally, putting the code all together in one program, we get:

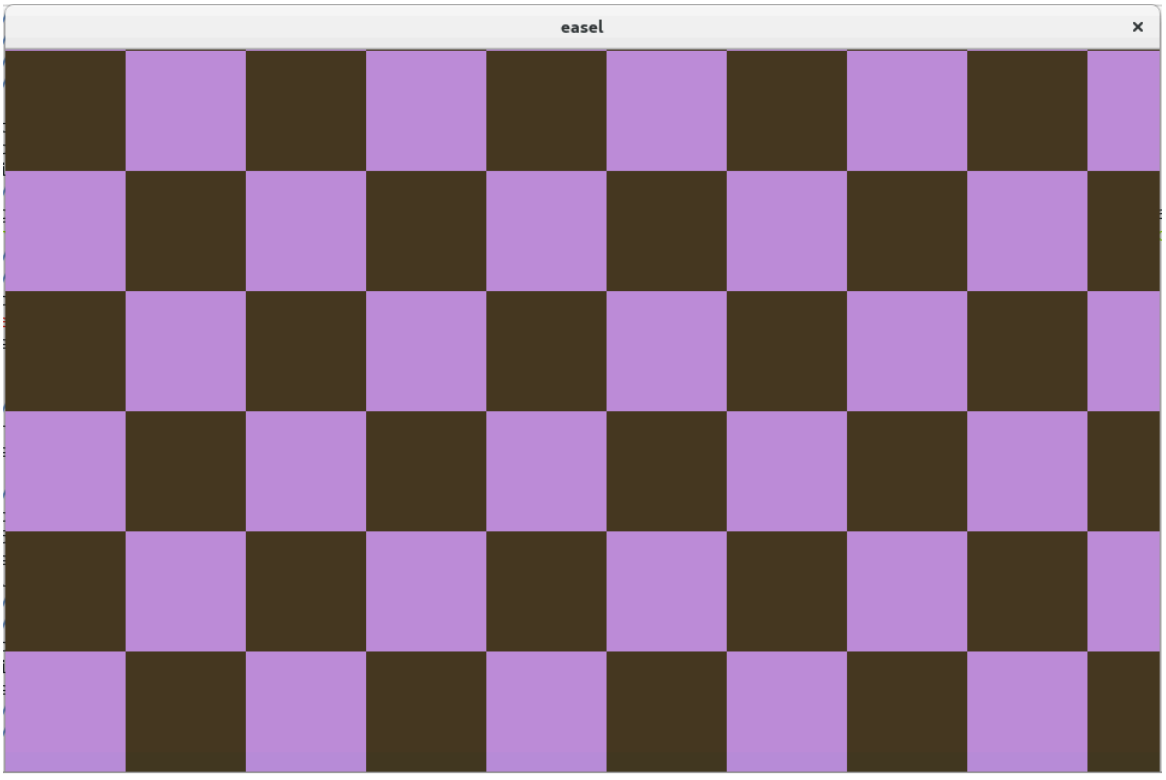
```
/*squares.es*/
pix col1 = #BF8FD600;
pix col2 = {70, 58, 39, 170};
pix canvas[960][720];
int W = 960, H = 720;

function void plaid(int p_w) {
    int x,y;
    for (y = 0; y < H; y++) {
        for (x = 0; x < W; x++) {
            if (((x/p_w)%2) == ((y/p_w)%2))
                canvas[x][y] = col1;
            else
                canvas[x][y] = col2;
        }
    }
}

plaid(100);
draw(canvas, 0, 0);
```

After saving your program (and assuming you've already run make in both the easel/ directory and glwrap/ directory), run these commands and the following output should appear on your screen.

```
$ ./ecc.sh squares.es
$ ./squares
```



3. Language Reference Manual

3.1 Keywords, Comments, and Whitespace

3.1.1 Keywords

The following are keywords reserved by the language and are not permitted as variable names.

bool	false	true	int	pix	red	alpha	tan	randos
float	function	if	else	void	green	cos	log	
return	for	while	draw	draw_size	blue	sin	rando	

3.1.2 Comments and Whitespace

Comments within a program are ignored and delineated by
`/*[text]*/`
Hence, all comments must have an opening "/*" and a closing "*/", which is then skipped by the parser.

There must be at least one space between tokens in order to separate them unless a separator (",", ".", or ";") is between them. A given token is therefore not allowed to have whitespace within it, lest it be interpreted as two separate tokens. Otherwise, whitespace is ignored.

3.2 Identifiers and Data Types

3.2.1 Names

Names are used to identify variables of data types as well as functions. They are allowed to be a sequence of alphanumeric characters and underscores, excepting those that match with reserved keywords, which will be interpreted as keywords in accordance with the grammar rules to follow. A name must begin with a letter or underscore, and upper and lower-case letters are not considered equivalent.

These are examples of names accepted by easel:

```
sampleName
Sample_name2
_sampleName_3
```

And the following name will be rejected:

```
3sample
```

3.2.2 Primitive Data Types

3.2.2.1 Primitive Variables

Primitive variables can be of the following 4 data types:

`int`	32-bit (or default machine architecture size) 2's complement number values
`float`	32-bit

`bool` 1-bit true of false values. Can be represented using “true” or “false” keywords or simply “1” or “0” are acceptable

`pix` 32-bit integer values. A pix stores a color value from 0-16777216 plus and alpha value from 0-255. Therefore, the color and alpha values represent blocks of bits.

Integers are signed 32-bit values and may take defined values from -268435456 to 268435455. Any values outside of this range will compile, but the integer will take on an undefined value.

Integers may be used anywhere floats can be used and conversion is automatic. That said, a float value cannot necessarily be used anywhere an int value is expected.

3.2.2.2 Primitive Literals

easel supports the following primitive literal types: integer literal, floating point literal, boolean literal, and pix literal. Integer literals are either written in decimal notation or hexadecimal notation. Decimal notation is defined as one or more digits between 0 and 9. Hex notation is denoted by a “#” followed by one or more numbers or letters (either uppercase or lowercase are accepted).

Following are examples of integer literals:

```
9
199
#09aeff
4aae00 /* this fails because without the '#' it is not interpreted as a hex value */
```

Floating point literals in decimal form consist of an integer value followed by an optional decimal point and other integer value (representing the fraction) as well as an optional exponent. If the first integer value does not exist, then there must be a decimal point followed by an integer value. An exponent may follow either the first integer value, or the decimal and second integer value, or both. No hex values are permitted in this representation.

Following are examples of floating point literals:

```
1.0
-2e15
0.34e99
42 /* this is not a floating point literal (rather it's an int literal) */
```

Boolean literals are simply true or false and correspond to integer values of 1 or 0. Boolean values in general will be promoted to integer or float status if placed into the context of mathematical comparisons.

Pix literals are primarily represented by four consecutive integer expressions. These take the form of four unsigned 8-bit integer values in “list notation” which represent the red, green, blue, and alpha values of the pix respectively[1].

[1] The red, green, and blue values indicate the amount of each respective color is in the pixel, while the alpha value stores information regarding transparency. A lower alpha value means the pixel will be more transparent when blended

Any valid integer literal or expression can be used for each of the list values. However, each value will be treated as an unsigned 8-bit integer. It is therefore advised that a programmer use integer expressions in the pix literal list that evaluate to proper 8-bit unsigned integer values, as other values may compile but behave in an undefined manner.

Additionally, 32-bit int literals, in either decimal or hexadecimal form, may also be used in the context of pixel values.

The following are examples of pixel literals:

```
{255, #ff, 0, #98}
#ff29ae00
{300, #gg, 257, -5} /* rejected because #gg is not a valid hex value */
{300, #ff, 257, -5} /* would compile but behave in an undefined manner because
                       the fourth value is signed rather than unsigned */
```

3.2.3 Non-Primitive Data Types

Variables storing a reference to a piece of data (rather than the data itself) and built using the above-mentioned primitive data types are created using the following types:

`array`	a data structure to store a sequence of adjacent values of a given data type.
`matrix`	a data structure to store a sequence of adjacent arrays of a given data type. Only 2-dimensional matrices are supported in easel.
`function`	a reference to a function instance.

Functions are able to be referred to as literals in their own right as well. A function literal, or anonymous function, is defined in the following form:

```
function type (function-parameter-list) {statements}
```

Function literals, or anonymous functions, can be passed as parameters to other functions (see section 3.6).

3.3.1 Simple Variable Declarations

Primitive variables are declared simply by providing the variable's type followed by its name. Declaring a variable without assigning it to a value with automatically initialize the variable with a "0"[2]. Definition can happen within a separate statement, or at the same time as declaration. The only exception is when function calls are required: a function call performed at the same time as declaration will result in undefined behavior.

---

with other pixels, whereas a higher alpha value will be more opaque. Alpha values are unimportant in the current implementation of easel, as only one drawing context may be made at a time; in future versions, these values will play a larger role in blending different drawings.

[2] The default value of zero allows a user to easily create and initialize a "canvas" (or matrix of pixels) in one go. Such a matrix can be drawn immediately if desired without the need for explicit initialization.

```
int var; /* declares variable of type int called var; var currently equals 0 */
var = 5; /* var now equals 5 */
int var2 = 10; /* create a new variable, var2, and set it equal to 10 */
```

3.3.2 Array Declarations

Arrays are declared using the type the array will store followed by the name of the variable and bracket notation. Values within brackets must be integer expressions, and a size must be provided at declaration in order for the declaration to be valid. When an array is declared, all elements in the array will be initialized to 0. Unlike simple variables, an array cannot be declared and defined on the same line.

```
pix arr[5]; /* declares an array of pixels with length 5 */
pix brr[]; /* this will fail with a parsing error because a size must be provided */
```

Matrices are created by nesting two arrays. Therefore, the syntax is similar although you must specify the size of both the inner and outer arrays. To do this, the programmer must declare the type of the matrix, the name, and then use two sets of brackets. The first bracket specifies the inner array (the number of columns) and the second bracket specifies the out array (the number of rows).

```
float matrix[4][8]; /* decalres a matrix with 8 rows and 4 columns */
int matrix[][3]; /* this fails because the size of both the inner and outer array must be
                specified */
```

3.3.3 Scope

Scope is best explained with a brief description of program structure. A given program consists of the global space, wherein any statement can be executed, or a function can be defined.

The scope of a variable in easel is defined to be within the function in which it is declared. Regions within control or loop statements have access to all other variables declared within the same function.

Global variables can be accessed and modified by any functions within the program.

3.4 Expressions

Expressions can take many forms but at their essence they return mathematical values. In their simplest form, expressions give the value of primitive literals or variables of primitive data types.

Expressions that are parenthesized will be evaluated first:

```
1 + (2 + 3) = 1 + 5
```

3.4.1 Unary operators on numeric values

Integer and floating point values are able to be given specific signs, positive or negative, by preceding them with a unary operator “-”.
Additionally, boolean values can utilize the unary operator “!” to flip its values (i.e. !true will return false). Using the “!” operator on an integer that is not equal to 0 is not allowed and will throw a compile time error.

Additionally, the increment ("++") and decrement ("--") operators can be used on integers, floating point numbers, and pix variables to either increase the current value by 1 ("++") or decrease the current value by 1 ("--").

3.4.2 Postfix Expressions: element access, property access, and function calls

After evaluating the simple expressions mentioned above, any postfix operators are applied (including all of the unary operators).

Of key importance for easel's functionality is the array subscript operator ("[*ind*]"). A programmer can both assign array elements and access elements using this operator.

```
int arr[5];
arr[2] = 10;
arr[2]; /* returns 10 */
```

Matrices additionally use the subscript operator. In easel, a matrix is stored as [*col*][*row*] so to access a particular element, use two consecutive subscript operators.

```
int matrix[5][5];
matrix[2][0] = 7;
matrix[2][0]; /* returns 7 */
matrix[1][1]; /* returns 0 because arrays/matrices are initialized to 0 */
```

Arrays additionally have a "." operator for property access. The "." is used to get the size of the outer array (i.e. for a matrix, the number of rows), using the keyword "size".

```
pix arr[10], matrix[2][5];
arr.size; /* returns 10 */
matrix.size; /* returns 5 */
```

Pix data types also use the "." operator. Property access is used for pixels to both access the individual color attributes ("red", "green", "blue", and "alpha") and to assign them to new values.

```
pix p = {255, #ff, 0, #aa};
p.red; /* returns 255 */
p.green; /* returns #ff */
p.blue; /* returns 0 */
p.alpha; /* returns #aa */

p.red = 0;
p; /* returns {0, #ff, 0, #aa} */
```

Finally, function calls are expressed using the function's name followed by the postfix expression "()". Within the parentheses, a user may provide a list of the required parameters.

Below is an example of defining a function called “foo” with arguments “int x”, “int y”, and “float z”, which returns the value of “x + y + z” when the function is called.

```
function float foo(int x, int y, float z) {
    return x + y + z;
}

foo(5, 6, 7.0) /* returns 18.0 */
```

3.4.3. Mathematical Expressions

Mathematical expressions are evaluated in the standard order of operations: parentheses (seen earlier), exponents, multiplication, division, addition, and subtraction. These operators are left-to-right associative as in standard mathematics.

easel supports the following mathematical operations (in order of precedence):

Operation	Symbol	Example
Exponentiation	^	`2^3 /* returns 8 */`
Multiplication	*	`2*3 /* returns 6 */`
Division	/	`6/2 /* returns 3 */`
Addition	+	`2+3 /* returns 5 */`
Subtraction	-	`3-2 /* returns 1 */`

3.4.3. Equivalence and Comparison

In easel expressions are compared before being tested for equality and they are evaluated left-to-right. easel supports the follow relational operators (listed in order of precedence).

Operator	Symbol	Example
Less than	<	`4 < 5 /* returns true */`
Greater than	>	`4 > 5 /* returns false */`
Less than or equal to	<=	`4 <= 4 /* returns true */`
Greater than or equal to	>=	`5 >= 4 /* returns true */`
Equals	==	`4 == 4 /* returns true */`
Not Equal	!=	`3 != 3 /* returns false */`

Functions, whether anonymous or named, cannot be compared and will return an error (this is also invalid syntax as functions are not considered expressions in easel).

3.4.4. Logical Expressions

Boolean values can be evaluated using the logical operators AND ("&&") and OR ("||"). In logical expressions, "&&" is given precedence. In the case of numbers as described above, all values not equal to 0 are considered "TRUE" bool values and all values equal to 0 are considered "FALSE".

3.4.5. Assignment

Assignment occurs by setting a variable on the left-hand side of an "=" to the value on the right-hand side of the "=". In addition to numeric literals, a variable can additionally be set to the value of any valid mathematical expression.

```
int a = 10;
float b = 4.0 + 5.1;
float c;
a + b = c; /* fails with a parsing error; a variable being assigned must be on left side */
```

Pix values may be assigned to ints and vice versa. However, neither int nor pix values can be assigned to floats and float values cannot be assigned to pix or int variables. This is meant to discourage assignment of incompatible types to prevent loss of information-it may be overridden using "return" statements in a function as a primitive cast (see section 3.6.1).

3.5. Statements

A general statement can be a declaration (which may include an expression), an expression by itself, control statements, and loop statements for iterative processes. Declarations and expressions must be followed by a semicolon. Control statements (if and if-else) and loop statements (for and while) do not need to be followed by a semicolon.

3.5.1. If Statements

If statements provide a condition which evaluates to true or false. If true, a sequence of statements in a region (bracketed on the left and right sides) will execute. If the expression is false, the region is skipped and the next statement will proceed to execute.

If statements have the following form:

```
if ( boolean-expression ) { statements }
```

```
int x, y;
x = 10;
y = 20;
if (x < y) {
    x = x + 15;
}
```

If the expression in parentheses is not a boolean expression, the program will *not compile* as an explicit boolean expression must be used.

If the parentheses are empty, the compiler will provide a parsing error and the program will not compile.

3.5.2. If-Else Statements

If-else statements offer an additional path following evaluation of the expression: if the boolean-expression evaluates to true, the first statement region is executed and the second is ignored. If the expression evaluates to false, the second statement region is executed and the first is ignored. Following either of these two events, the next statement following the if-else statement will proceed to execute.

If-else statements take the following form:

```
if ( boolean-expression ) { statements } else { statements }
```

```
int a = 10;
bool b = true;
if (b) {
    a = a + 2;
}
else {
    a = a - 2;
}
```

3.5.3. For Loops

For statements begin with an expression (meant to declare and initialize an iterator variable), run a particular region of statements, evaluate another expression (meant to be a form of iterator) and then check to see if a given condition is true (usually if the iterator has reached a certain threshold).

For loops take the following form:

```
for ( expression1; expression2; expression3 ) { statements }
```

```
int i, int canv[10];
for (i = 0; i < 10; i++) {
    canv[i] = 255;
}
```

3.5.4. While Loops

While statements first check to see if a given expression evaluates to true and if so, it executes the region. The expression is then evaluated at the conclusion of every iteration and if it continues to be true, the region continues to be executed. When the expression evaluates to false, the region is skipped and the following statement is executed.

While loops have the following form:

```
while ( boolean-expression ) { statements }
```

```
int a = 0;
while (a < 10) {
    a++;
}
```

As in control statements (if and if-else), the boolean expression within parentheses must be a boolean-expression or the program will not compile.

3.6. Functions

Functions form the foundation of easel's functionality. A few important things to note:

1. Primitive elements are passed by value, while functions and arrays are passed by reference.
2. Functions may return a void data type, but void is not a valid data type for any other value.
3. Functions may be passed as arguments to other functions, either as a function literal / anonymous function or simply as a named entity.
4. Functions may be called recursively, and can call any other functions defined in the global scope.
5. If two functions within the global scope have the same name, the compiler will throw an error.

3.6.1. Function Declarations and Definitions

When declared as part of the global statement flow of a program, functions must be defined at declaration.

Function declarations have the following form:

```
function type name ( function-parameter-list )
```

Only primitive types (i.e. bool, pix, int, and float) and void are considered valid return types. Functions and arrays/matrices are not allowed as return types.

The function parameter list is composed of declarations, including declarations of other functions. However, when passing arrays as arguments, the declaration is syntactically structured slightly differently than statement declarations. Brackets are added following array's type rather than after the identifier:

```
function void userarr(int[] arr) {
    return 0;
} /* this is a valid function */

function void usearr(int arr[]) {
    return 0;
} /* this function is invalid and will throw a parsing error */
```

Matrices can also be passed into functions-however, the number of columns for any parameter matrix must be declared within the function as an integer literal. This means that even variable names with integer values are not permitted within a function declaration.

```
function void usemat1(int[10][] mat) {
    /* body */
} /* this is a valid function */

function void usemat2(int[][] mat) {
    /* body */
} /* invalid; this will throw a parse error */

int j = 10;
function void usemat3(int[j][] mat) {
    /* body */
} /* invalid; this will throw a parse error */
```

To define a function as a parameter of another function, the following syntax is used for the parameter:
function *return-type* (*parameter-type list*) *function_name*

```
function void takes_func(function int (int, int) passed_func, int x, int y) {
    passed_func(x,y);
}
```

In the above example, the void function takes_func accepts three arguments:

1. A function that returns an integer and takes two integers as parameters (passed_func)
2. An integer x
3. An integer y

takes_func then calls passed_func with arguments x and y.

As mentioned in a previous section, primitive values are passed by value, whereas arrays, matrices, and functions are passed by reference. The most important implication of this is that the values stored in arrays and matrices may be changed within a function call.

```
function void userarr(int[] firstarr, int l) {
    int i = 0;
    for (i; i < l; i++) {
        firstarr[i] = 4;
    }
}
int myarr[10]; /* default values set to 0 */
usearr(myarr, 10); /* all myarr values are now 4 */
```

The function definition is composed of a series of statements and optionally ends with the keyword "return" and an expression. Functions are only able to return primitive values. Functions are defined as:
{ *statements* **return** *expression* }
or
{ *statements* }

If a return statement is not included, the function will return a value of "0", cast to the function's return type, by default.

```
function float test(float a, float b) {
    a = a + 1.2;
    b = b - 2.7;
    return a + b;
}
```

An important note regarding the function's return type: numeric types can return values of any other numerical types. That is, a function returning a float value with a return statement returning an int expression.

```
function float testa(int a) {
    return a;
}
testa(3); /* returns 3.0 */

function int testb(float a) {
    return a;
}
testb(3.4);
testb(3.5);
testb(3.6); /* all return 3 */

function float testc(pix a) {
    return a;
}
testc({#ff, #ff, #ff, #ff}); /* returns #ffffffff00000000 or -1.0 */
```

This can be exploited as a primitive form of casting.

A function can only be declared without definition when the function itself is being passed as an argument to another function.

```
function void my_func(int x, int y, function int (int w, int z) operator) {
    /* body */
}
```

The "operator" function is thus declared as a parameter. Any function passed as the operator parameter must have a matching return type and argument list or the program will be rejected by the compiler. Note the syntactic difference with normal function declarations- the function id is the last portion of the declaration.

3.6.2. Anonymous Functions

Functions may also take the form of function literals (AKA anonymous functions). These functions differ from the standard form in that they are not named.

Anonymous functions follow the format:
function *type* (*function-parameter-list*) { *statements* **return** *expression* }

```
function pix (bool b) { return !b; }
```

3.6.3. Invoking Functions

Functions can be called anywhere during statement flow. A function is called by using its name and passing it arguments as defined by the function declaration.

A function call follows the following format:
function_name (*argument-list*);

```
function int add(int x, int y) {
    return x + y;
}
int a = add(3,2); /* a now equals 5 */
```

To pass another function to a function the function's name can be provided as a parameter.

```
function void my_func(int x, int y, function int (int, int) operator) {
    /* body */
}
function int pass(int x, int y) {
    return x + y;
}
my_func(3,4,pass); /* returns 7 */
```

The above will pass the function "pass" as a parameter to my_func. Alternatively, an anonymous function can be passed to my_func. The functionality below is equivalent to the example above.

```
my_func(3, 4, function int (int x, int y){return x+y;});
```

3.6.4. Built-In Functions

Whereas many other languages have some mechanism of printing strings, easel's built-in method of output is drawing a canvas to the monitor.

easel has two different functions for drawing out to the screen. The signature of the first draw function responsible for the drawing is:
draw(pix name[][], int x, int y);

The arguments are a 2-dimensional pixel matrix as well as an x and y value that determines where the top left corner of the canvas window is placed. An easel program is ultimately about creating this matrix of pixels using various functions or coding prowess to create a visual representation on screen. The draw function will look up the matrix passed to it in the symbol table in order to determine screen size.

```
pix canv[900][1000];
draw(canv, 0, 0); /* this will draw a black canvas in the top left corner of the screen */
```

The other function used for drawing in easel is defined as:
draw_size(pix name[][], int w, int h, int x, int y);

The arguments are a 2-dimensional pixel matrix[3] as well as an x and y value that determines where the top left corner of the canvas window is placed. Additionally, there are w and h values that specify the width and height of the canvas to be drawn out to the screen. Note that the width and height passed to the function must match the width and height of the matrix passed or a user's program will crash.

Note: Draw can only be called once and should be the final statement in your program! Calling draw multiple times will result in undefined behavior. A user executing a program that calls either of the draw functions can end the program when the window containing the created image is closed.

easel also offers a number of mathematical functions, including:
float tan(float x)
float sin(float x)
float cos(float x)
float log(float base, float value)
which allow the user to calculate more complicated geometric functions than the standard arithmetic operators.

easel has a random number generator, which generates a floating point value between 0 and 1 inclusive. This value can be multiplied, added, or subtracted to in order to provide a random value within a desired range.
float rando()

Finally, easel also has a random number generator that allows you to specify a seed.
float randos(int s)

4. Project Plan

At the beginning of the semester, we met as a group once per week. We initially had meetings scheduled on either Monday's or Wednesday's directly after class (depending on other scheduling conflicts). Shortly after we began having our weekly TA meetings on Monday mornings, we found it more useful to have our group meetings directly after the TA meeting while the new goals were still fresh in our minds and to maximize the amount of time for each party to work on a newly assigned task. Additionally, as the semester progressed and deadlines were approaching, we began to meet twice a week; these meetings were generally on weekends where

[3] This as well as the function draw_size are the only functions that take matrices without explicitly declared column sizes.

we would meet in a library for approximately four to five hours and have a pseudo-hackathon where team members were allowed to come and go as necessary.

We prioritized our tasks and goals based on deadlines. During our preliminary meetings, we brainstormed many ideas that we thought would be cool to implement and as the process went on, we cut ideas that were not actually feasible in the given amount of time for this project - it is much easier to have to many ideas and scale back than have too simple of a language and need to scramble to add more features.

We started our scanner, parser, and abstract syntax tree as early as possible. We tried to concurrently write the AST and language reference manual, thus ensuring that our language was unambiguous and that our LRM was consistent with our AST. Because the core of our language is similar to MicroC, we were able to get started rather easily by building off of the MicroC files. We additionally began creating tests for checking that our AST was correct by starting to write our test suite as soon as the initial AST, scanner, and parser files were completed. Once we began adding new features to the language, such as the semantic checker and codegen, we already had a basic test suite to begin building off of, which also helped streamline the process. We did run into the issue of not starting our semantic checker early enough to finish it before the Hello World milestone, so to meet our deadlines we worked through the first steps in codegen to create a Hello World program and then spent the remainder of the semester working simultaneously on our semantic checker and codegen. For these tasks, we essentially split the team up so half of the team members focused on one file while the other half work on the other; this split helped speed up writing because certain team members became experts on one file while the other team members became experts on the other.

Although our team members had clear roles, we all contributed to the code. To coordinate these contributions, we kept in constant contact through our Facebook group chat; our goal of this was to avoid stepping on anyone's toes or two people working on implementing the same feature simultaneously. We kept track of progress and changes by creating a GitHub repository. To avoid merge conflicts, all team members forked the code from Oswin's repo and then created pull requests so that he had final say in which files got merged and he could confirm as the system architect that everything was as it should be. By communicating and giving one team member final say in what gets merged, we kept all files organized and avoided any potential merge conflicts.

4.1 Programming Style Guide

OCaml:

- Keep lines of code to approximately 120 characters max
- For nested functions, indent line two spaces
- If a series of let or and statements are on top of each other (and all relating to the same functionality), no additional indents are needed
- In match statements, the -> goes on the same line as the "match" phrase. The case matching begins on the next line, four spaces in (starting with the name of the condition to be matched). On the next line, the | begins two spaces in, with a space following the | so that the second condition is in line with the first condition (and so on for all following cases to match)
- A blank newline is entered between different functions that are connected by an "in"
- Only fully functioning code should be pushed to the master branch
- Use descriptive but short variable names where possible

- Use underscores and lowercase letters for naming -- if the code is slightly altering one particular variable, name the new variable the same name with a ‘ following (i.e. let var = … in let var’ = …)
- Follow indentation styles of the section you’re adding to (even if it’s not correct) for consistency and readability
- Comment out code that you believe is superfluous (or incorrect) if you did not write that code and have no confirmation that the code is not needed

easel:

- Indent four spaces for functions and control flow
- Wrap (or split up) lines that are longer than approximately 120 characters
- For control flow, put opening brace on same line as statement (i.e. if (true) {) and match indentation of closing brace with start of statement
- Declare functions and variables with descriptive names
- Declare multiple variables on the same line (where possible)
- Name variables with lowercase letters and underscores

4.2 Project Timeline

9/12: Team formed
9/14: First brainstorming meeting (looked into a few different ideas but did not settle on a topic yet)
9/21: Roles assigned, final language idea agreed on, and langauge ideas fleshed out
9/28: Submitted our language proposal (we didn’t want to do too much work on any part of our language before feedback from the proposal)
10/3: First TA meeting -- used this feedback to begin writing our LRM
10/12: First draft of our LRM uploaded by Tyrus for review by the team. Oswin created our GitHub repo and began the scanner, parser, and AST
10/26: Final changes to the LRM completed by this date and the LRM is submitted. During this time, continued working on AST, scanner, and parser to finalize ideas and remain consistent with the LRM
11/5: Began working on the semantic checker (semant.ml) and fail cases. During this time, worked on semantic checker, modified our LRM based on TA feedback, and wrote some test cases
11/16: Began working on codegen. Implemented the features needed for Hello World to run
11/20: Hello World program compiles and runs (without semantic checking). Also added new output files to our test suite to ensure that our code was properly compiling down to LLVM IR on files other than Hello World (it was!). We spent the next few weeks working both independently and in meetings to get the rest of codegen and the semantic checker finished (working based on TODO’s in the code and removing them when a task was implemented)
12/11: Our intended demo test program (the mandelbrot) is able to be rendered
12/15: Finished the semantic checker
12/16: Finished codegen and merged the two files onto the same branch. Dealt with problems that arose there are fixed bugs while simultaneously working on the report and presentation
12/19: Presentation - all code and presentation slides must be completed

4.3 Roles and Responsibilities

Although we all had specific roles, those positions were used more as a guideline of who to ask questions to when stuck (for example, we went to Tyrus when we had questions about language semantics and Oswin with problems about the code) and who had final say should any conflicts arise about language functionality (although we had no real issues with this other than Oswin reminding us that many features are not as easy to implement in OCaml as they were to brainstorm).

Manager	Danielle Crosswell
Language Guru	Tyrus Cukavac
System Architect	Yuan-Chao (Oswin) Chou
Tester	Xiaofei (Sophie) Chen

4.4 Software Development Environment

- OCaml, with Ocamlyacc and Ocamllex for compiling the scanner and parser
- The LLVM Ocaml library: used within our ocaml programs to generate LLVM IR code
- LLC: used to compile generated LLVM code to unix object code
- GCC used to link compiled easel code without OpenGL wrapper (to provide graphics output)
- Bash script to create the easel compiler collection, and automate compilation and linking to a final easel program
- Ubuntu 16.04 Virtual Machine (provided by the TA's): Use of the VM ensured all team members were compiling with the same tools and in the same OS environment; additionally, unix environment helped streamline the development and debugging process
- OpenGL libraries as an API for graphics production
- Vim and Sublime: each team member used his/her own prefered text editor
- Git: used for version control
- GitHub: A centralized remote repository which allowed team members to work on different branches while maintaining the integrity of the master branch of code

4.5 Project Log

Our GitHub commit log, excluding merge commits.

```
2016-12-19  b2f2e5e  sophiechan - play with mandelbrot
2016-12-19  e5bbb72  sophiechan - operations on different types
2016-12-19  c492a73  sophiechan - checking different literal types
2016-12-18  2f79b4b  sophiechan - modify test cases
2016-12-18  294e844  sophiechan - adding anonymous function test
2016-12-18  8c64988  Yuan Chao Chou - Merge pull request #59 from
                     sophiechan/dev
2016-12-18  2aa4ce7  Yuan Chao Chou - Merge pull request #54 from
                     dcrosswell/dev
2016-12-18  61f497c  Danielle Crosswell - created new julia set demo
```

```
2016-12-18  62a43c0  sophiechan - remove arraylit
2016-12-18  3572cea  sophiechan - modify array test cases and ast
2016-12-18  2a9fa53  Oswin Chou - Add codegen to Makefile
2016-12-18  08195e1  sophiechan - modify fail cases
2016-12-18  72694ac  sophiechan - modify fail cases
2016-12-18  b2fa488  sophiechan - unrecognized function failure
2016-12-18  5f30cb0  Oswin Chou - We assign variable with expression
                     of the same type in easel
2016-12-18  54f5883  sophiechan - modify fail cases
2016-12-18  1175b5b  Oswin Chou - Fix a test case output
2016-12-18  4a3c654  Oswin Chou - build_fneg for floats
2016-12-18  9f38862  Oswin Chou - Build another statement for
                     assigning initializer to global var
2016-12-18  7e12ace  Yuan Chao Chou - Merge pull request #56 from
                     thc2125/dev
2016-12-18  641fd07  Tyrus Cukavac - Fixed a parse issue that allowed
                     an array to be declared without an integer inside
                     brackets
2016-12-18  bcc7f4f  sophiechan - modify fail cases
2016-12-18  9f0e847  Tyrus Cukavac - Fixed an issue with "EleAt"
                     wherein float array indices were allowed to
                     compile
2016-12-18  a6858b7  sophiechan - modify fail cases
2016-12-18  9f90914  sophiechan - modify fail cases
2016-12-18  05fc580  Tyrus Cukavac - Fixed semant to allow for
                     assignment of int values to floats
2016-12-18  172b969  sophiechan - unrecognized function failure
2016-12-18  2649ebd  Tyrus Cukavac - Added printb function for
                     booleans; fixed a semant error that allowed for n-
                     dimensional matrices
2016-12-17  92ce123  Oswin Chou - Use property assignment in
                     mandelbrot_anon.es
2016-12-17  a29bfad  Oswin Chou - Support specifying window position
2016-12-17  ecb8420  Oswin Chou - Move samples for demo to the demo
                     folder
2016-12-17  246b82b  Oswin Chou - Trivial refinement for autotest.sh
2016-12-17  c516009  Oswin Chou - Add -o option and usage message for
                     ecc.sh
2016-12-17  d4ed6ce  Yuan Chao Chou - Merge pull request #55 from
                     sophiechan/dev
2016-12-16  858adf0  sophiechan - modify ast pretty printing
2016-12-16  2096aa3  sophiechan - adding ast
2016-12-16  ec3ffa3  sophiechan - modifying functions in tests
2016-12-16  57beeda  Danielle Crosswell - fixed hello world program
2016-12-16  15b4f18  Danielle Crosswell - run ecc.sh with -c or no
                     options
2016-12-16  9b0fc94  sophiechan - adding ast files
2016-12-16  c63d0a4  Oswin Chou - Correct the output for tests/test-
                     arr-args1.out
2016-12-16  9bb18ac  sophiechan - negate pix values
2016-12-16  48ca4b7  sophiechan - modifying fail cases
2016-12-16  450faa9  Oswin Chou - 0 for fib(0)
```

```
2016-12-16  b566ddb  Oswin Chou - Flag functions that have been
                     checked
2016-12-16  74b24e3  Oswin Chou - Fix some test cases
2016-12-16  1a6b3ab  Tyrus Cukavac - Debug messages removed so
                     compiler should work
2016-12-16  397a115  Tyrus Cukavac - Fixes to semant; Mandelbrot_anon
                     now compiles; 5-argument draw function is now
                     called "draw_size"
2016-12-16  17a83bb  Oswin Chou - Patch print statements for
                     tests/test-arith.es
2016-12-16  ea380cf  Oswin Chou - Merge branch 'tyrant-dev' into dev
2016-12-16  923108d  Oswin Chou - ArrLit is removed
2016-12-16  70acc18  Oswin Chou - Merge branch 'codegen' into dev
2016-12-16  ff5bcc8  Oswin Chou - Fix an API inconsistency problem
2016-12-16  bbb8448  Oswin Chou - Merge branch 'dev' of
                     https://github.com/OswinC/easel into dev
2016-12-16  29cfd4f  Yuan Chao Chou - Merge pull request #48 from
                     dcrosswell/codegen
2016-12-16  8035282  Yuan Chao Chou - Merge pull request #47 from
                     sophiechan/dev
2016-12-16  67e47d1  Yuan Chao Chou - Merge pull request #46 from
                     sophiechan/dev
2016-12-16  47a814b  Tyrus Cukavac - Fixed warnings in semant.ml
                     except for arrlit based warnings
2016-12-16  c5be9bc  Oswin Chou - Fix AST inconsistency problems
                     caused by merging codegen & dev
2016-12-16  0af17ed  Danielle Crosswell - commented out ArrLit and
                     removed warning messages
2016-12-16  9266c49  sophiechan - add print functions to function
                     table
2016-12-16  4249f4f  Danielle Crosswell - added changing pixel value
                     using property access
2016-12-16  c15afde  sophiechan - remove unused function
2016-12-16  90ff103  sophiechan - exhaust all pattern matching
2016-12-16  8bb1c5f  Oswin Chou - Unary mul, div, pow ops are removed
2016-12-16  b4851f0  Danielle Crosswell - working on prooacc for
                     assign
2016-12-16  ba0ceb2  Oswin Chou - Merge branch 'codegen' into dev
2016-12-16  d9f1457  Oswin Chou - Add anonymous version of the
                     mandelbrot sample code
2016-12-16  11cc12b  Oswin Chou - Mandelbrot sample code using named
                     function
2016-12-16  ff5b157  Oswin Chou - Revert an experimental change
2016-12-16  f58047f  Oswin Chou - Some initial works for returning
                     arrays
2016-12-16  2c6bbb0  Oswin Chou - Support building anonymous functions
2016-12-15  bb108aa  Yuan Chao Chou - Merge pull request #45 from
                     thc2125/dev
2016-12-15  ee64741  Oswin Chou - Revise the mandelbrot sample program
2016-12-15  b911c52  Oswin Chou - Fix a problem for passing array into
                     functions
2016-12-15  1873246  Tyrus Cukavac - Might have made a mistake while
```

```
                    merging
2016-12-15  7b228c4 Tyrus Cukavac - Semant fully functional; all
                    TODOs complete (pending additional requirements);
                    arraylits removed
2016-12-15  c3948cf Danielle Crosswell - fixed declaration on same
                    line as init
2016-12-15  5bcdadb Danielle Crosswell - removed some warnings in
                    codegen
2016-12-15  7f7bcad Tyrus Cukavac - Next pass through on checking
                    functions; may be reverting some issues as far as
                    final ordering of check
2016-12-15  79c65f1 Yuan Chao Chou - Merge pull request #43 from
                    dcrosswell/codegen
2016-12-15  5b925db Oswin Chou - Assign return name by judging on the
                    return type
2016-12-15  fdb0e16 Danielle Crosswell - removed unary mult, div, and
                    pow
2016-12-15  5597a0b Danielle Crosswell - debugged pix property access
                    and added size operator
2016-12-15  91ab3f5 Tyrus Cukavac - More work done on function
                    semantic checking; arrays are working properly,
                    but now anonymous functions passed as formals need
                    to be dealt with
2016-12-15  639e3f1 Danielle Crosswell - declare pix on same line as
                    init
2016-12-14  29af96d Oswin Chou - Support passing arrays as parameters
2016-12-14  3bf7303 Oswin Chou - Remove a duplicate of expression
                    building in building return
2016-12-14  86c8d7f Oswin Chou - Support building local variables of
                    array types
2016-12-14  3cd2155 Oswin Chou - Code formating
2016-12-14  a6f20dc Oswin Chou - Change the naming of Arr to ArrRef
2016-12-14  6a2bf0c Oswin Chou - Revise the format for declaring
                    arrays in the formal
2016-12-14  bcebc01 Oswin Chou - Change naming of Arr to ArrRef
2016-12-14  33d49c8 Tyrus Cukavac - Nearing completion of function
                    semantic checking; issues with formal types
                    sticking if type is array or matrix
2016-12-14  514ed1d Tyrus Cukavac - First pass at function call
                    semantic checking; not functioning haha
2016-12-14  3e00a80 Tyrus Cukavac - Minor change to unary op checking
2016-12-14  702e425 Tyrus Cukavac - Added the pow, the two random
                    functions, and the updated draw function names to
                    semantic check
2016-12-14  b8b513c Tyrus Cukavac - ArrLit Semantic Checking now
                    working as well
2016-12-14  05b5c8b Tyrus Cukavac - EleAt is now working with regards
                    to arrays of arrays-still needs indices
2016-12-14  32ec099 Tyrus Cukavac - First pass at prop access
2016-12-14  93de37d sophiechan - check_assign for semantic check
2016-12-14  b2c27cc sophiechan - assign part for semantic check
2016-12-14  22b5ae4 sophiechan - array literal for semantic check
```

```
2016-12-14  b2d5074  sophiechan - arraylit semantcheck
2016-12-14  20f343a  sophiechan - modify semantic check
2016-12-14  540b602  Oswin Chou - Add singleton for checking system
2016-12-14  0341b38  Oswin Chou - Add semant.ml and testall.sh into
                     Makefile
2016-12-14  58e2c75  Oswin Chou - Use function signature to check and
                     build function table
2016-12-14  121bbf4  Oswin Chou - Revise the format for declaring
                     arrays in the formal
2016-12-13  1649a94  Oswin Chou - Remove junk comments
2016-12-13  fef388c  Oswin Chou - Minor changes for mandelbrot.es
2016-12-13  4dc5fac  Oswin Chou - Support passing functions as
                     parameters
2016-12-11  9a6d425  Tyrus Cukavac - ArrLit Semantic Checking now
                     working as well
2016-12-11  1d88579  Tyrus Cukavac - First pass at prop access
2016-12-11  874a230  Oswin Chou - Update Pix literal usage
2016-12-11  d81df95  Oswin Chou - Revise mandelbrot programs
2016-12-11  69530be  Oswin Chou - Two refinements in codegen
2016-12-11  ae1dab8  Oswin Chou - Add alpha channel into Pix literal
2016-12-11  cd07970  sophiechan - check_assign for semantic check
2016-12-11  7bb4bb7  sophiechan - assign part for semantic check
2016-12-10  3b3607b  Yuan Chao Chou - Merge pull request #38 from
                     thc2125/codegen
2016-12-10  b1cadc8  Tyrus Cukavac - Added random numbers and
                     logarithms to codegen
2016-12-10  ad643ef  Tyrus Cukavac - Merged with work from 12/9
2016-12-09  0e27672  Danielle Crosswell - started PropAcc functions
2016-12-09  92fcb75  Oswin Chou - Remove finished todos
2016-12-09  6f4576a  Oswin - Merge pull request #36 from
                     sophiechan/master
2016-12-09  93e6a8c  Oswin - Merge pull request #35 from
                     dcrosswell/codegen
2016-12-07  330e0b9  Danielle Crosswell - declare and initialize vars
                     in the same line
2016-12-05  2ceeae7  Danielle Crosswell - removed warning messages for
                     A.Call
2016-12-05  513a8c2  Danielle Crosswell - added trig functions
2016-12-05  41b6157  Oswin Chou - Add an easel sample code for drawing
                     a mandelbrot
2016-12-05  09679b5  Oswin Chou - Add a test case for subtracting a
                     float from an int
2016-12-05  2b06665  Oswin Chou - Support type promotion for binary
                     operations
2016-12-05  e585a81  Oswin Chou - Add test case for return type
                     promotion
2016-12-05  56a4584  Oswin Chou - Support return int by float and
                     return float by int
2016-12-05  7138418  Oswin Chou - Support data type promotion for
                     multiply operation
2016-12-05  35368e5  Oswin Chou - Reformat the byte-wise arrangement
                     of pix
```

```
2016-12-05  0c38217  Oswin Chou - Support pow of float to float
2016-12-05  6dbf478  Oswin Chou - Improve hello.es
2016-12-05  6872ec5  Danielle Crosswell - added if statements and
                     fixed .out file
2016-12-05  ca63214  Oswin Chou - Merge branch 'dcrosswell-codegen'
                     into codegen
2016-12-05  fc23867  sophiechan - Delete the main function
2016-12-04  0fc8ed7  Danielle Crosswell - unary operators A.pow
2016-12-04  da57aa2  Oswin Chou - Support basic function invoking
2016-12-04  f6008c6  sophiechan - array literal
2016-12-04  dc04765  sophiechan - array literal for semantic check
2016-12-04  67ca2bb  sophiechan - Delete the main function
2016-12-04  5d40cf0  Oswin Chou - Support unary addition
2016-12-04  bd1f57f  Oswin Chou - Support for loop and add some
                     refinement
2016-12-04  c7f1e26  Tyrus Cukavac - Not and Neg unary operators
                     functioning
2016-12-04  664a78d  Tyrus Cukavac - All binary operators functioning
                     (unstable, noncomprehensively tested)
2016-12-04  a5d2389  Tyrus Cukavac - Not and Neg unary operators
                     functioning
2016-12-04  2f0dd36  Oswin - Merge pull request #28 from
                     thc2125/codegen
2016-12-04  f2e5ea4  Tyrus Cukavac - First pass at binops; merged with
                     DC's assignment code
2016-12-04  3bbe99a  Tyrus Cukavac - First pass at basic binops
2016-12-04  171ec03  Oswin - Merge pull request #27 from
                     thc2125/codegen
2016-12-04  6eab491  Oswin - Merge pull request #26 from
                     dcrosswell/codegen
2016-12-04  a2e4eab  Oswin Chou - Fix bug- should use the resulting
                     builder for building termination block
2016-12-04  c75e88d  Oswin Chou - Add codegen code for while statement
2016-12-04  0167aa5  Tyrus Cukavac - Minor tweak to line 190
2016-12-04  d222c6f  Danielle Crosswell - added array assignment and
                     access
2016-12-04  46ab5b6  Tyrus Cukavac - Added support for pixel and float
                     assignment
2016-12-04  d647de3  Danielle Crosswell - fixed codegen merging
                     conflicts
2016-12-04  0f71b09  sophiechan - modify semantic check
2016-12-04  18dea95  Danielle Crosswell - started func declarations,
                     while loop, cleaned up code
2016-12-02  cd1ada6  Danielle Crosswell - worked on while and func
                     body
2016-11-29  cecfcbf  Oswin - Merge pull request #23 from
                     dcrosswell/codegen
2016-11-28  5582cb7  Danielle Crosswell - started working on array
                     assignment
2016-11-28  41e8519  Danielle Crosswell - working on array assignemnts
2016-11-28  762a1b6  Oswin Chou - Don't have to manually number the
                     register naming
```

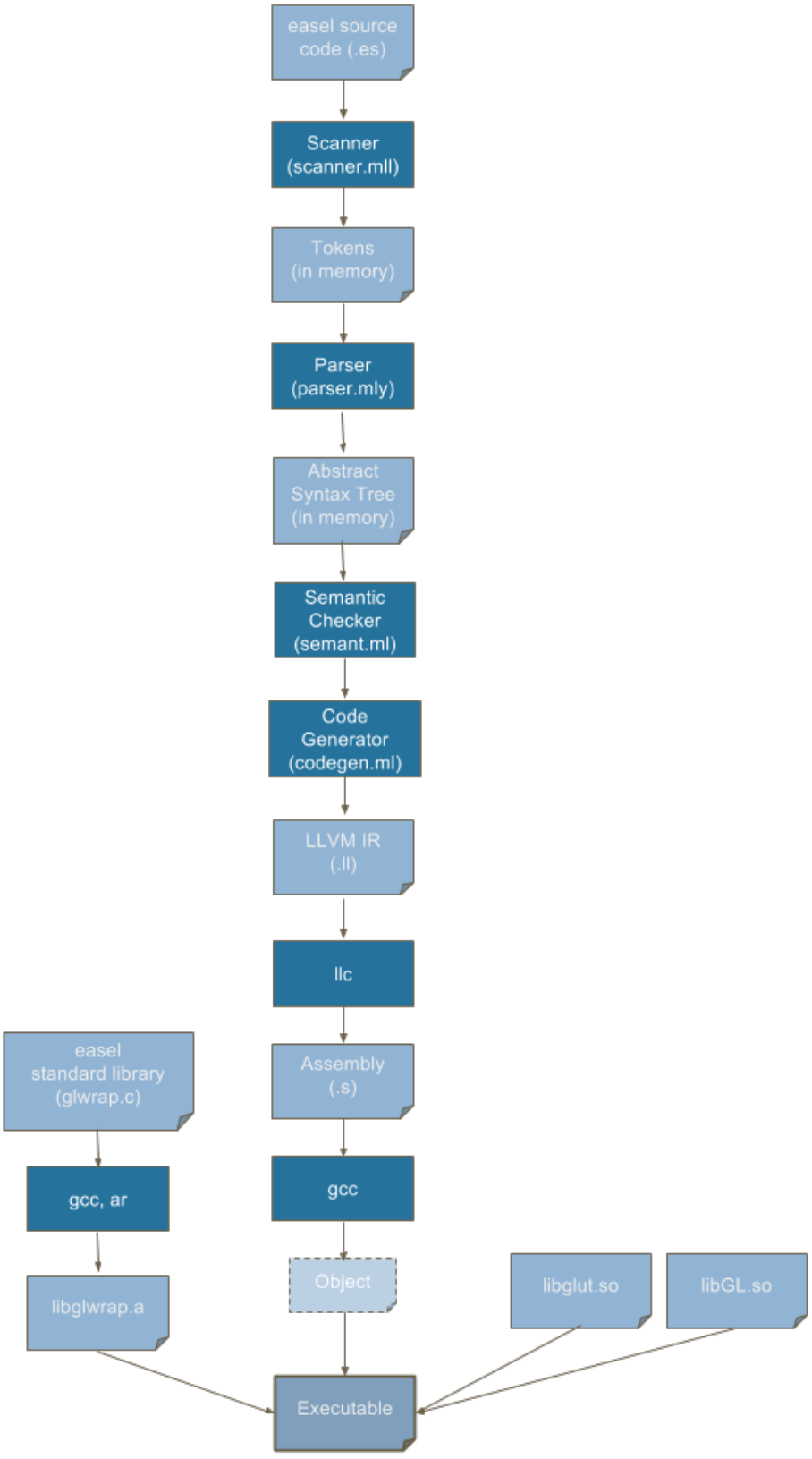
```
2016-11-28  ea3bf2c  Oswin Chou - Support building and looking up
                     local symbol table
2016-11-22  6a477f4  Danielle Crosswell - started assignment in
                     codegen
2016-11-22  2429dd9  sophiechan - Add the output files for test cases
2016-11-22  877dc2a  Oswin - Merge pull request #19 from
                     dcrosswell/codegen
2016-11-22  d0d716a  Danielle Crosswell - remove .ll and .out files on
                     make clean
2016-11-22  f86aa11  Oswin Chou - Minor typos
2016-11-22  b4b342f  Oswin Chou - Compile before start testing
2016-11-22  d0bf17d  Oswin - Correct a new section formating for
                     README.md
2016-11-22  f884b82  Oswin Chou - More formating for README.md
2016-11-22  af3b678  Oswin Chou - Modify README.md and format it by
                     the Github Markdown syntax
2016-11-22  73ad2f5  Tyrus Cukavac - modified autotest with ecc
                     compiling 'hello'
2016-11-22  c53e79e  Tyrus Cukavac - Added a few new instructions for
                     README; formatted markdown slightly
2016-11-22  364c571  Oswin Chou - Enable compilation tests
2016-11-22  3db1f94  Oswin Chou - Add print function and two test
                     cases
2016-11-22  edc31d3  Danielle Crosswell - fixed codegen merge issues
2016-11-22  6530ccc  Oswin Chou - Add comments for explaining the code
                     in hello.es
2016-11-22  c8c189f  Oswin Chou - Add and modify glwrap apps
2016-11-22  b78d5f1  Oswin Chou - Support passing a matrix into draw()
2016-11-21  1a817dd  Oswin Chou - Statements should be reversed to get
                     the correct order
2016-11-21  f896359  Oswin Chou - Add clean rule for glwrap/Makefile
2016-11-21  4ac02da  Danielle Crosswell - Merge remote-tracking branch
                     'upstream/codegen' into codegen
2016-11-21  0ce8c1c  Oswin Chou - Add hello.ll for temporalily testing
                     purpose
2016-11-20  47f7058  Oswin Chou - Add declaration for do_draw(...)
2016-11-20  d1a3d0c  Oswin Chou - Rename ltype_* to lltype_* so not
                     confusing with left value
2016-11-20  e274d2f  Oswin Chou - Make dumbhello.es less dumb
2016-11-20  9fd12b2  Oswin Chou - Support global declaration and array
                     types
2016-11-20  259322a  Oswin Chou - Remove initializer from the simple
                     hello world
2016-11-20  feb85e7  sophiechan - simple script for hello world
                     program
2016-11-20  1e13a78  Oswin Chou - Remove libglwrap.a from the top
                     directory
2016-11-20  2f07476  Oswin Chou - Copy libglwrap.a to the top folder
                     after compiled
2016-11-20  f8b38ce  Tyrus Cukavac - Some additions to codegen.ml-
                     early stages of stmts
2016-11-20  21b2bbb  Tyrus Cukavac - Tweaked pixel literals slightly
```

```
                    so its a little more structured
2016-11-20  1650147  Oswin Chou - Remove a debug message
2016-11-20  9428559  Oswin Chou - Shut off debug messages in glwrap.c
                    with macros
2016-11-20  d6b22bb  Oswin Chou - Simplify apps/hello.ll
2016-11-20  ed1eee5  Oswin Chou - Add Makefile for glwrap
2016-11-19  6fc392c  Oswin Chou - Rename opengl/ to glwrap/
2016-11-19  dd534f4  Oswin Chou - Reorganize glwrap and add hello.ll
                    for calling glwrap
2016-11-19  2f5eddd  Oswin Chou - Rename testall.sh to autotest.sh
2016-11-19  bd740f2  Oswin Chou - Reorganize the folder structure of
                    tests/
2016-11-19  6968760  Oswin Chou - Add wrapper for opengl and an app
                    that uses it
2016-11-19  b8e1469  Danielle Crosswell - modify codegen and easel for
                    hello world
2016-11-19  b08eeaa  Oswin - Merge pull request #12 from
                    thc2125/codegen
2016-11-19  cc38899  Tyrus Cukavac - 1st attempt at getting basic
                    codegen working
2016-11-19  ae4b119  Oswin - Merge pull request #10 from
                    dcrosswell/codegen
2016-11-19  6ffed7a  Danielle Crosswell - arrays in codegen
2016-11-19  465163e  Danielle Crosswell - added array functionality to
                    codegen
2016-11-19  d8ea412  Oswin Chou - Add hello world sample code
2016-11-19  ca93c66  Danielle Crosswell - created codegen file and
                    started expressions
2016-11-16  969c983  Oswin Chou - [] are used for arrays in easel
2016-11-16  5fa9502  Danielle Crosswell - modified .err files for some
                    fail cases
2016-11-16  5f1ae69  Danielle Crosswell - created .err files for fail
                    cases and fixed bugs in some files
2016-11-16  dc93517  Danielle Crosswell - started fail cases for
                    assignment and expressions
2016-11-09  4e28d35  Danielle Crosswell - modified .err files for some
                    fail cases
2016-11-09  3e30018  Danielle Crosswell - created .err files for fail
                    cases and fixed bugs in some files
2016-11-08  31c67aa  Oswin Chou - Add singleton for checking system
2016-11-08  4e161b2  Oswin Chou - Add semant.ml and testall.sh into
                    Makefile
2016-11-08  073983e  Oswin Chou - Use function signature to check and
                    build function table
2016-11-08  602199f  Oswin Chou - Add code for testing fail cases
2016-11-07  6ed89bc  Oswin Chou - "%%" operator is not supported in
                    easel
2016-11-07  a5f22a2  Tyrus Cukavac - No more mod_expr-conforms to
                    mult_exprs
2016-11-07  6169763  Tyrus Cukavac - Added minor modulus test in
                    test.es
2016-11-07  c0cb76e  Tyrus Cukavac - Add mod operator to scanner
```

```
                    parser
2016-11-07  b2494df  Tyrus Cukavac - Added grammar reference for
                    convenience
2016-11-07  1fe63b3  sophiechan - Adding arithmatic ast tests and
                    float ast tests
2016-11-01  b2542cf  Danielle Crosswell - created semant.ml
2016-11-01  acf8497  Oswin Chou - Ignore spaces and newlines for the
                    diff in testall.sh
2016-11-01  c9baa51  Oswin Chou - Merge branch 'sophie-master'
2016-11-01  62cac3e  Oswin Chou - Fix some test cases
2016-11-01  b3856d9  Oswin Chou - Merge branch 'master' of
                    git://github.com/sophiechan/easel into sophie-
                    master
2016-11-01  1415621  Oswin Chou - Fix .gitignore
2016-11-01  f93fc39  sophiechan - modify the cirfun ast
2016-11-01  639c7c6  Oswin Chou - Add test-floatlit1.ast for test-
                    floatlit1.es
2016-11-01  de8f088  sophiechan - add test ast
2016-11-01  5287506  Oswin Chou - Only ignore .ast and .diff at top
                    directory
2016-11-01  240d2c0  Oswin - Merge pull request #2 from
                    sophiechan/master
2016-11-01  3216027  sophiechan - adding test cases
2016-10-31  6167135  Tyrus Cukavac - Merge branch 'master' of
                    https://github.com/OswinC/easel
2016-10-31  c73dd15  Oswin Chou - Minor changes in README
2016-10-31  75d1201  Oswin Chou - Add README.md
2016-10-30  638ec82  Oswin Chou - Add anonymous version of the sqaure
                    example
2016-10-30  c22e2b2  Oswin Chou - Initial framework of testing system
2016-10-30  f1b1f3a  Oswin Chou - Refine pretty printing
2016-10-30  e3ccc39  Oswin Chou - Add a sample code for drawing a
                    square
2016-10-30  a48f628  Oswin Chou - Parse pix literals and remove some
                    epsilon rules
2016-10-30  5a50b05  Oswin Chou - Some initial work for parsing n-ple
                    literal
2016-10-30  b9d419e  Tyrus Cukavac - Removed my test text file
2016-10-30  d916e36  Tyrus Cukavac - Added grammar reference for
                    convenience
2016-10-30  ff3c2ae  Tyrus Cukavac - test
2016-10-30  ce748d0  Oswin Chou - Parse array literals
2016-10-30  a4655b6  Oswin Chou - Parse hexadecimal literals
2016-10-30  5fd54dd  Oswin Chou - Don't need these precedence level
                    assignments
2016-10-30  f87456a  Oswin Chou - Allow assign_expr expand to
                    assign_expr
2016-10-30  6f1a521  Oswin Chou - Parse the dot property accessor
2016-10-30  02af60d  Oswin Chou - Reverse back the order of
                    formals/aformals on assignment
2016-10-30  f1d339e  Oswin Chou - Refine the pretty printing for
                    anonymous functions
```

```
2016-10-30  af171b0  Oswin Chou - Left-associative is correct for
                     exp_expr
2016-10-30  cbb1451  Oswin Chou - Fix a trivial typo
2016-10-29  268d260  Oswin Chou - Parse declaration of anonymous
                     function
2016-10-29  0c9e353  Oswin Chou - Use Ocaml's "and" keyword to allow
                     mutual reference
2016-10-29  a9adb9f  Oswin Chou - Add a test case for parsing function
                     as a formal
2016-10-29  27762bf  Oswin Chou - Parse anonymous function type
2016-10-29  fc39f7e  Oswin Chou - Add parsing rules for expressions
2016-10-29  04a3e0d  Oswin Chou - Remove *.output for make clean
2016-10-29  6b57dd8  Oswin Chou - Parse the arrays in function
                     signature
2016-10-29  9de3bec  Oswin Chou - Support parsing array declaration
2016-10-27  07c387f  Oswin Chou - Refinement for printing formals
2016-10-26  35ed002  Oswin Chou - Parse float literals
2016-10-26  c6b37f3  Oswin Chou - Make the code parsing bool less
                     redundant
2016-10-26  fb0e0a6  Oswin Chou - Parse float and pix keywords
2016-10-26  5a14532  Oswin Chou - Add comments in the easel testing
                     file
2016-10-26  d33257d  Oswin Chou - Support parsing variable
                     definition/declaration
2016-10-25  31ebb9f  Oswin Chou - Support nested comment
2016-10-25  c942206  Oswin Chou - Remove tailing whitespaces
2016-10-25  490a66a  Oswin Chou - Add OpenGL sample code
2016-10-12  40cdeac  Oswin Chou - Ignore built files
2016-10-12  7e02625  Oswin Chou - Add an easel source file for testing
2016-10-12  af7f978  Oswin Chou - Parse the keyword "function" for
                     recognizing a function
2016-10-12  ac0c506  Oswin Chou - Refine the printing output for AST
2016-10-12  ebacc66  Oswin Chou - Support putting statements in the
                     global scope
2016-10-12  2b6eed1  Oswin Chou - The story of easel starts here
```

5. Architectural Design



5.1. Top Level (easel.ml)

"easel.ml" marks the starting point of the compiler. It defines the different compiler options "-a" for Ast, "-l" for LLVM IR, and -"c" for compile. If no options are provided to easel.native, then the compile option is selected by default. Immediately following a determination of compiler options, a program (provided to easel by stdin) is scanned and parsed. The program is then semantically checked to ensure there are no mismatched types in assignment, function calls, or within expressions. Depending on the user-provided option, the compiler will do the following:

-a: the AST option essentially takes the abstract syntax tree and re-prints it in a program-like representation.

-l: The LLVM option will generate the LLVM code, utilizing codegen (outputting to stdout on unix machines). The LLVM IR string will not be checked for any irregularities.

-c: The Compile option will generate the LLVM IR code and then check it for any irregularities. The Code will then be output on stdout.

5.2. Scanner (scanner.mll)

The scanner will take a Lexer buffer read from a given ".es" file by the Lexing module and tokenize it into the various:

Operators (+, -, *, /, etc.)
Literals (int, pix, float, bool, function)
Separators (brackets, braces, semi-colons, commas, etc.)

Comments (nested and otherwise) and whitespace are removed from the program as well, leaving a purely tokenized program. These tokenized values are then passed to the parser.

5.3. Parser (parser.mly) and Abstract Syntax tree (ast.ml)

The parser (parser.mly) is given a tokenized program and builds the abstract syntax tree as defined by ast.ml. The primary separators are semicolons (";") and braces ("{}") which are used to break the program into statements and blocks respectively. Within statements are declarations, control statements, loop statements, and expressions, which are composed of various mathematical/numerical operators. Blocks are used to define the scopes of local variables, the bodies of control statements and loops, as well as functions.

Functions are represented in OCaml by a record which contains information about the function's return type, name, formals (i.e. parameters), body, and a boolean value to indicate whether or not it has been checked by the semantic checker. Ultimately, a list of function declarations and a list of global statements are created and represent the final program.

5.4. Semantic Checker (semant.ml)

The above-mentioned lists of function declarations and global statements are then passed to the semantic checker. It takes the function list and converts it to a map of strings which utilizes each function's name as its key. The semantic checker then passes through the list of global statements. It ensures any variable definitions occur between matching (or acceptably matching) types while also raising errors if it finds any invalid types (such as void). Moreover, any declared variables are added to a hash table of global functions. Expressions within statements, such as those utilizing binary operators or property access variables such as "red" or "size", are also checked to ensure these operations are only performed on the appropriate types. Assignment

operations (those utilizing a "=" symbol) also ensure that a given variable exists in the global hash table, otherwise an error is raised and compilation stops.

While passing through global statements, the semantic checker also identifies any function calls. If a function is called, the semantic checker looks for it in the list of function declarations, raising an error if it does not exist. If it is in the list of declarations, the semantic checker proceeds to check the function. This is done by first ensuring all formals (parameters) are valid types to be passed, then by checking the statements within the function's body. As variables are declared within a function body, they are added to a map of strings holding local variables and their given type. This list is checked (in addition to the global hash table) upon encountering any assignments within the function body.

Of particular interest during the semantic check of functions is the analysis of anonymous functions used as formals. These, like traditional variable formals, are added to a list of "local functions". This list is now checked in addition to the list of global function declarations when a function call occurs within the function.

All statements within a function body are examined in the same manner. If a function passes the check without any type mismatches, that functions "checked" variable will be marked as true to prevent a later redundant function check. Once all statements have been checked, any functions that were not called by the global environment are also checked to ensure semantic consistency. If an error is raised, compilation stops, even though the function is unused. This is to ensure that only well-formed programs are compiled by easel.

5.5. Code Generation (codegen.ml)

The code generator plays the role of translating the given abstract syntax tree into LLVM IR. The entire building process is performed by iterating through the statements, which construct functions. Expressions are the basic building blocks of easel, and they are evaluated for generating the LLVM IR code, and return the corresponding LLVM address. Variable declarations are defined as a subset of statements, so variables are built and stored in symbol tables during the iteration of building statements.

Because statements are allowed to be put in the global scope (thus so-called global statements) in easel, and the entry point of an easel program is the first global statement, we create a built-in main function for each easel program and wrap the global statements in the main function as the entry for the executable. The statements of the main function and programmer-defined functions are then iterated through and the LLVM IRs are generated accordingly. During the building process, anonymous functions are defined, named by a string composed of a reserved keyword (____ReSeRvEd_AnOnYmOuS_fUnC__) and a sequential number. They are then collected in another map upon matching the anonymous function's AST. This map is then iterated through by applying the same process after the main and named functions are built.

Any variables declared within a block are local variables, which live until the end of the block, and are replaced by the local variables declared in the sub-blocks with the same name during the lifetime of the later-defined variable. To cope with this dynamic environment, we take advantage of the immutable nature of OCaml's Map. Additionally, we utilize the fact that the entry and the exit of a block flag the start and the end of the lifetime of the local symbol table to manage local variables with the fewest lines of code as possible. Upon entering a block, the symbol table of its parent block is passed in and referenced for creating a new symbol table whenever it evaluates a variable declaration statement. This newly created symbol table is then returned as the

basis for declaring variables in following statements. Upon leaving this block, the symbol table is simply discarded and the parent block's symbol table stay unchanged.

Finally, the created LLVM module is translated to a string of LLVM IR and the string is printed as the output of easel.native.

5.6. The easel compiler collection (ecc.sh) and the OpenGL wrapper (glwrap.c)

easel is dependent on external, low-level functions in order to produce the screen images for which the language was made. We chose to use OpenGL as an API to produce the images. Consequently, the easel.native compiler will produce the LLVM code representing the primary easel program, but an additional linker is required to link this program with the functions used in our OpenGL wrapper (written in C).

The "ecc.sh" script automates this entire process, taking as arguments options for the easel.native compiler as well as the filepath to the ".es" file to be compiled. "ecc.sh" runs easel.native with the given option -a, -l, or -c (the default if no options are given). Given the -a option, the AST string produced by easel.native will be output to a file with the given filename and extension ".ast". Utilizing the -l option prompts easel.native to generate the LLVM string into a file with the given filename and extension ".ll". Finally, the -c option generates a checked version of the LLVM string which is then output as a ".ll" file. This file is compiled into assembly code (".s") by the LLVM compiler LLC, and then compiled to an object and linked with the compiled "glwrap" library, which defines the draw functions used by easel. ecc.sh removes the leftover LLVM and assembly files, leaving an executable with the same base filename as that of the original .es file.

5.7. Implementation Roles

Code	Author(s)
Makefile in easel/	Oswin, Danielle, Sophie
Makefile in glwrap/	Oswin
scanner.mll	Oswin, Tyrus
parser.mly	Oswin, Tyrus, Danielle
ast.ml	Oswin, Tyrus, Danielle, Sophie
semant.ml	Oswin, Tyrus, Sophie
codegen.ml	Oswin, Danielle, Tyrus
easel.ml	Oswin, Danielle, Sophie, Tyrus
autotest.sh	Oswin, Sophie
ecc.sh	Sophie, Danielle
glwrap.c	Oswin, Tyrus

6. Test Plan

6.1 Source Language and Generated Programs

```
/* mandelbrot_anon.es */
pix canvas[960][840];
int W = 960, H = 840;

function pix[960][] graph(pix[960][] canv, int w, int h, function pix (int, int) painter) {
    int x, y;
    for (y = 0; y < h; y++) {
        for (x = 0; x < w; x++) {
            canv[x][y] = painter(x, y);
        }
    }
    return canv;
}

function int red(int x, int y) {
    float a = 0., b = 0., c, d, n = 0.;
    while ((c = a * a) + (d = b * b) < 4. && n++ < 880.) {
        b = 2. * a * b + y * 8e-9 - .645411;
        a = c - d + x * 8e-9 + .356888;
    }
    return 255 * (((n - 80.)/800.) ^ 3.);
}

function int green(int x, int y) {
    float a = 0., b = 0., c, d, n = 0.;
    while ((c = a * a) + (d = b * b) < 4. && n++ < 880.) {
        b = 2. * a * b + y * 8e-9 - .645411;
        a = c - d + x * 8e-9 + .356888;
    }
    return 255 * (((n - 80.)/800.) ^ .7);
}

function int blue(int x, int y) {
    float a = 0., b = 0., c, d, n = 0.;
    while ((c = a * a) + (d = b * b) < 4. && n++ < 880.) {
        b = 2. * a * b + y * 8e-9 - .645411;
        a = c - d + x * 8e-9 + .356888;
    }
    return 255 * (((n - 80.)/800.) ^ .5);
}

draw_size(graph(canvas, W, H, function pix (int x, int y) {
    pix p = 0;
    p.red = red(x, y);
    p.green = green(x, y);
    p.blue = blue(x, y);
    return p;
}), W, H, 200, 0);
```

Figure 1: mandelbrot_anon easel code

```
; ModuleID = 'easel'

@canvas = global [840 x [960 x i32]] zeroinitializer
@W = global i32 0
@H = global i32 0

define i32 @blue(i32 %x, i32 %y) {
entry:
  %x1 = alloca i32
  store i32 %x, i32* %x1
  %y2 = alloca i32
  store i32 %y, i32* %y2
  %a = alloca double
  store double 0.000000e+00, double* %a
  %b = alloca double
  store double 0.000000e+00, double* %b
  %c = alloca double
  store double 0.000000e+00, double* %c
  %d = alloca double
  store double 0.000000e+00, double* %d
  %n = alloca double
  store double 0.000000e+00, double* %n
  br label %while

while:                                            ; preds = %while_body, %entry
  %a19 = load double, double* %a
  %a20 = load double, double* %a
  %tmp21 = fmul double %a19, %a20
  store double %tmp21, double* %c
  %b22 = load double, double* %b
  %b23 = load double, double* %b
  %tmp24 = fmul double %b22, %b23
  store double %tmp24, double* %d
  %tmp25 = fadd double %tmp21, %tmp24
  %tmp26 = fcmp olt double %tmp25, 4.000000e+00
  %n27 = load double, double* %n
  %n28 = load double, double* %n
  %tmp29 = fadd double %n28, 1.000000e+00
  store double %tmp29, double* %n
  %tmp30 = fcmp olt double %n27, 8.800000e+02
  %tmp31 = and i1 %tmp26, %tmp30
  br i1 %tmp31, label %while_body, label %merge

while_body:                                       ; preds = %while
  %a3 = load double, double* %a
  %tmp = fmul double 2.000000e+00, %a3
  %b4 = load double, double* %b
  %tmp5 = fmul double %tmp, %b4
  %y6 = load i32, i32* %y2
  %tmp7 = sitofp i32 %y6 to double
  %tmp8 = fmul double %tmp7, 8.000000e-09
  %tmp9 = fadd double %tmp5, %tmp8
  %tmp10 = fsub double %tmp9, 6.454110e-01
  store double %tmp10, double* %b
  %c11 = load double, double* %c
  %d12 = load double, double* %d
  %tmp13 = fsub double %c11, %d12
```

```
  %x14 = load i32, i32* %x1
  %tmp15 = sitofp i32 %x14 to double
  %tmp16 = fmul double %tmp15, 8.000000e-09
  %tmp17 = fadd double %tmp13, %tmp16
  %tmp18 = fadd double %tmp17, 3.568880e-01
  store double %tmp18, double* %a
  br label %while

merge:                                            ; preds = %while
  %n32 = load double, double* %n
  %tmp33 = fsub double %n32, 8.000000e+01
  %tmp34 = fdiv double %tmp33, 8.000000e+02
  %tmp35 = call double @pow(double %tmp34, double 5.000000e-01)
  %tmp36 = fmul double 2.550000e+02, %tmp35
  %tmp37 = fptosi double %tmp36 to i32
  ret i32 %tmp37
}

define i32 @green(i32 %x, i32 %y) {
entry:
  %x1 = alloca i32
  store i32 %x, i32* %x1
  %y2 = alloca i32
  store i32 %y, i32* %y2
  %a = alloca double
  store double 0.000000e+00, double* %a
  %b = alloca double
  store double 0.000000e+00, double* %b
  %c = alloca double
  store double 0.000000e+00, double* %c
  %d = alloca double
  store double 0.000000e+00, double* %d
  %n = alloca double
  store double 0.000000e+00, double* %n
  br label %while

while:                                            ; preds = %while_body, %entry
  %a19 = load double, double* %a
  %a20 = load double, double* %a
  %tmp21 = fmul double %a19, %a20
  store double %tmp21, double* %c
  %b22 = load double, double* %b
  %b23 = load double, double* %b
  %tmp24 = fmul double %b22, %b23
  store double %tmp24, double* %d
  %tmp25 = fadd double %tmp21, %tmp24
  %tmp26 = fcmp olt double %tmp25, 4.000000e+00
  %n27 = load double, double* %n
  %n28 = load double, double* %n
  %tmp29 = fadd double %n28, 1.000000e+00
  store double %tmp29, double* %n
  %tmp30 = fcmp olt double %n27, 8.800000e+02
  %tmp31 = and i1 %tmp26, %tmp30
  br i1 %tmp31, label %while_body, label %merge

while_body:                                       ; preds = %while
  %a3 = load double, double* %a
```

```
  %tmp = fmul double 2.000000e+00, %a3
  %b4 = load double, double* %b
  %tmp5 = fmul double %tmp, %b4
  %y6 = load i32, i32* %y2
  %tmp7 = sitofp i32 %y6 to double
  %tmp8 = fmul double %tmp7, 8.000000e-09
  %tmp9 = fadd double %tmp5, %tmp8
  %tmp10 = fsub double %tmp9, 6.454110e-01
  store double %tmp10, double* %b
  %c11 = load double, double* %c
  %d12 = load double, double* %d
  %tmp13 = fsub double %c11, %d12
  %x14 = load i32, i32* %x1
  %tmp15 = sitofp i32 %x14 to double
  %tmp16 = fmul double %tmp15, 8.000000e-09
  %tmp17 = fadd double %tmp13, %tmp16
  %tmp18 = fadd double %tmp17, 3.568880e-01
  store double %tmp18, double* %a
  br label %while

merge:                                            ; preds = %while
  %n32 = load double, double* %n
  %tmp33 = fsub double %n32, 8.000000e+01
  %tmp34 = fdiv double %tmp33, 8.000000e+02
  %tmp35 = call double @pow(double %tmp34, double 7.000000e-01)
  %tmp36 = fmul double 2.550000e+02, %tmp35
  %tmp37 = fptosi double %tmp36 to i32
  ret i32 %tmp37
}

define i32 @red(i32 %x, i32 %y) {
entry:
  %x1 = alloca i32
  store i32 %x, i32* %x1
  %y2 = alloca i32
  store i32 %y, i32* %y2
  %a = alloca double
  store double 0.000000e+00, double* %a
  %b = alloca double
  store double 0.000000e+00, double* %b
  %c = alloca double
  store double 0.000000e+00, double* %c
  %d = alloca double
  store double 0.000000e+00, double* %d
  %n = alloca double
  store double 0.000000e+00, double* %n
  br label %while

while:                                            ; preds = %while_body, %entry
  %a19 = load double, double* %a
  %a20 = load double, double* %a
  %tmp21 = fmul double %a19, %a20
  store double %tmp21, double* %c
  %b22 = load double, double* %b
  %b23 = load double, double* %b
  %tmp24 = fmul double %b22, %b23
  store double %tmp24, double* %d
```

```
  %tmp25 = fadd double %tmp21, %tmp24
  %tmp26 = fcmp olt double %tmp25, 4.000000e+00
  %n27 = load double, double* %n
  %n28 = load double, double* %n
  %tmp29 = fadd double %n28, 1.000000e+00
  store double %tmp29, double* %n
  %tmp30 = fcmp olt double %n27, 8.800000e+02
  %tmp31 = and i1 %tmp26, %tmp30
  br i1 %tmp31, label %while_body, label %merge

while_body:                                       ; preds = %while
  %a3 = load double, double* %a
  %tmp = fmul double 2.000000e+00, %a3
  %b4 = load double, double* %b
  %tmp5 = fmul double %tmp, %b4
  %y6 = load i32, i32* %y2
  %tmp7 = sitofp i32 %y6 to double
  %tmp8 = fmul double %tmp7, 8.000000e-09
  %tmp9 = fadd double %tmp5, %tmp8
  %tmp10 = fsub double %tmp9, 6.454110e-01
  store double %tmp10, double* %b
  %c11 = load double, double* %c
  %d12 = load double, double* %d
  %tmp13 = fsub double %c11, %d12
  %x14 = load i32, i32* %x1
  %tmp15 = sitofp i32 %x14 to double
  %tmp16 = fmul double %tmp15, 8.000000e-09
  %tmp17 = fadd double %tmp13, %tmp16
  %tmp18 = fadd double %tmp17, 3.568880e-01
  store double %tmp18, double* %a
  br label %while

merge:                                            ; preds = %while
  %n32 = load double, double* %n
  %tmp33 = fsub double %n32, 8.000000e+01
  %tmp34 = fdiv double %tmp33, 8.000000e+02
  %tmp35 = call double @pow(double %tmp34, double 3.000000e+00)
  %tmp36 = fmul double 2.550000e+02, %tmp35
  %tmp37 = fptosi double %tmp36 to i32
  ret i32 %tmp37
}

define [960 x i32]* @graph([960 x i32]* %canv, i32 %w, i32 %h, i32 (i32, i32)* %painter) {
entry:
  %canv1 = alloca [960 x i32]*
  store [960 x i32]* %canv, [960 x i32]** %canv1
  %w2 = alloca i32
  store i32 %w, i32* %w2
  %h3 = alloca i32
  store i32 %h, i32* %h3
  %painter4 = alloca i32 (i32, i32)*
  store i32 (i32, i32)* %painter, i32 (i32, i32)** %painter4
  %x = alloca i32
  store i32 0, i32* %x
  %y = alloca i32
  store i32 0, i32* %y
  store i32 0, i32* %y
```

```
  br label %while

while:                                            ; preds = %merge, %entry
  %y24 = load i32, i32* %y
  %h25 = load i32, i32* %h3
  %tmp26 = icmp slt i32 %y24, %h25
  br i1 %tmp26, label %while_body, label %merge27

while_body:                                       ; preds = %while
  store i32 0, i32* %x
  br label %while5

while5:                                           ; preds = %while_body6, %while_body
  %x18 = load i32, i32* %x
  %w19 = load i32, i32* %w2
  %tmp20 = icmp slt i32 %x18, %w19
  br i1 %tmp20, label %while_body6, label %merge

while_body6:                                      ; preds = %while5
  %canv7 = load [960 x i32]*, [960 x i32]** %canv1
  %y8 = load i32, i32* %y
  %canv9 = getelementptr inbounds [960 x i32], [960 x i32]* %canv7, i32 %y8
  %x10 = load i32, i32* %x
  %canv11 = getelementptr inbounds [960 x i32], [960 x i32]* %canv9, i32 0, i32 %x10
  %painter12 = load i32 (i32, i32)*, i32 (i32, i32)** %painter4
  %y13 = load i32, i32* %y
  %x14 = load i32, i32* %x
  %tmp = call i32 %painter12(i32 %x14, i32 %y13)
  store i32 %tmp, i32* %canv11
  %x15 = load i32, i32* %x
  %x16 = load i32, i32* %x
  %tmp17 = add i32 %x16, 1
  store i32 %tmp17, i32* %x
  br label %while5

merge:                                            ; preds = %while5
  %y21 = load i32, i32* %y
  %y22 = load i32, i32* %y
  %tmp23 = add i32 %y22, 1
  store i32 %tmp23, i32* %y
  br label %while

merge27:                                          ; preds = %while
  %canv28 = load [960 x i32]*, [960 x i32]** %canv1
  ret [960 x i32]* %canv28
}

declare i32 @draw_default(...)

declare i32 @do_draw(i32*, i32, i32, i32, i32, ...)

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

declare double @pow(double, double)

declare double @sin(double, ...)
```

```
declare double @cos(double, ...)

declare double @tan(double, ...)

declare double @log(double, ...)

declare double @rando()

declare double @randos(i32)

define i32 @main() {
entry:
  store i32 960, i32* @W
  store i32 840, i32* @H
  %H = load i32, i32* @H
  %W = load i32, i32* @W
  %tmp = call [960 x i32]* @graph([960 x i32]* getelementptr inbounds ([840 x [960 x i32]],
[840 x [960 x i32]]* @canvas, i32 0, i32 0), i32 %W, i32 %H, i32 (i32, i32)*
@____ReSeRvEd_AnOnYmOuS_fUnC__1)
  %cnvstmp = getelementptr inbounds [960 x i32], [960 x i32]* %tmp, i32 0, i32 0
  %H1 = load i32, i32* @H
  %W2 = load i32, i32* @W
  %do_draw = call i32 (i32*, i32, i32, i32, i32, ...) @do_draw(i32* %cnvstmp, i32 %W2, i32
%H1, i32 0, i32 0)
  ret i32 0
}

define i32 @____ReSeRvEd_AnOnYmOuS_fUnC__1(i32 %x, i32 %y) {
entry:
  %x1 = alloca i32
  store i32 %x, i32* %x1
  %y2 = alloca i32
  store i32 %y, i32* %y2
  %y3 = load i32, i32* %y2
  %x4 = load i32, i32* %x1
  %tmp = call i32 @red(i32 %x4, i32 %y3)
  %y5 = load i32, i32* %y2
  %x6 = load i32, i32* %x1
  %tmp7 = call i32 @green(i32 %x6, i32 %y5)
  %y8 = load i32, i32* %y2
  %x9 = load i32, i32* %x1
  %tmp10 = call i32 @blue(i32 %x9, i32 %y8)
  %tmp11 = mul i32 %tmp, 16777216
  %tmp12 = mul i32 %tmp7, 65536
  %tmp13 = mul i32 %tmp10, 256
  %tmp14 = add i32 %tmp11, %tmp12
  %tmp15 = add i32 %tmp14, %tmp13
  %tmp16 = add i32 %tmp15, 0
  ret i32 %tmp16
}
```

Figure 2: mandelbrot_anon LLVM IR code

```
/* jset.es */

pix canv[960][720];
int W = 960, H = 720;

function void graph(pix[960][] canvas, int w, int h, function pix (int, int) paint) {
    int x, y;
    for (y = 0; y < h; y++) {
        for (x = 0; x < w; x++) {
            canvas[x][y] = paint(x,y);
        }
    }
}

function float dim(int x) {
    float f = (x - H/2.0)/(H/2.0);
    return f;
}

function int red(int i, int j) {
    float x, y, X, Y, n;
    x = dim(i);
    y = dim(j);
    while (n<200 && (x*x + y*y)<1) {
        X = x*x;
        Y = y*y;
        x = X-Y + 0.36237;
        y = 2*x*y + 0.32;
        n++;
    }

    return log(2.718281, n) * 256;
}


function int green(int i, int j) {
    float x, y, X, Y, n;
    x = dim(i);
    y = dim(j);
    while (n<200 && (x*x + y*y)<1) {
        X=x;
        Y=y;
        x = X*X - Y*Y - 0.7;
        y = 2*X*Y+0.27015;
        n++;
    }

    return log(2.718281, n) * 96;
}

function int blue(int i, int j) {
    float x, y, X, Y, n;
    x = dim(i);
    y = dim(j);
    while (n<600 && (x*x + y*y)<1) {
        X=x;
        Y=y;
```

```
        x = X*X - Y*Y + 0.36237;
        y = 2*X*Y+0.32;
        n++;
    }

    return log(2.718281, n)*128;
}

function pix paint_jset(int x, int y) {
    return { red(x,y), green(x,y), blue(x,y), 0 };
}

graph(canv, W, H, paint_jset);
draw(canv, 0, 0);
```

Figure 3: jset easel code

```
; ModuleID = 'easel'

@canv = global [720 x [960 x i32]] zeroinitializer
@W = global i32 0
@H = global i32 0

define i32 @paint_jset(i32 %x, i32 %y) {
entry:
  %x1 = alloca i32
  store i32 %x, i32* %x1
  %y2 = alloca i32
  store i32 %y, i32* %y2
  %y3 = load i32, i32* %y2
  %x4 = load i32, i32* %x1
  %tmp = call i32 @red(i32 %x4, i32 %y3)
  %y5 = load i32, i32* %y2
  %x6 = load i32, i32* %x1
  %tmp7 = call i32 @green(i32 %x6, i32 %y5)
  %y8 = load i32, i32* %y2
  %x9 = load i32, i32* %x1
  %tmp10 = call i32 @blue(i32 %x9, i32 %y8)
  %tmp11 = mul i32 %tmp, 16777216
  %tmp12 = mul i32 %tmp7, 65536
  %tmp13 = mul i32 %tmp10, 256
  %tmp14 = add i32 %tmp11, %tmp12
  %tmp15 = add i32 %tmp14, %tmp13
  %tmp16 = add i32 %tmp15, 0
  ret i32 %tmp16
}

define i32 @blue(i32 %i, i32 %j) {
entry:
  %i1 = alloca i32
  store i32 %i, i32* %i1
  %j2 = alloca i32
  store i32 %j, i32* %j2
  %x = alloca double
  store double 0.000000e+00, double* %x
```

```
%y = alloca double
  store double 0.000000e+00, double* %y
  %X = alloca double
  store double 0.000000e+00, double* %X
  %Y = alloca double
  store double 0.000000e+00, double* %Y
  %n = alloca double
  store double 0.000000e+00, double* %n
  %i3 = load i32, i32* %i1
  %tmp = call double @dim(i32 %i3)
  store double %tmp, double* %x
  %j4 = load i32, i32* %j2
  %tmp5 = call double @dim(i32 %j4)
  store double %tmp5, double* %y
  br label %while

while:                                            ; preds = %while_body, %entry
  %n24 = load double, double* %n
  %tmp25 = fcmp olt double %n24, 6.000000e+02
  %x26 = load double, double* %x
  %x27 = load double, double* %x
  %tmp28 = fmul double %x26, %x27
  %y29 = load double, double* %y
  %y30 = load double, double* %y
  %tmp31 = fmul double %y29, %y30
  %tmp32 = fadd double %tmp28, %tmp31
  %tmp33 = fcmp olt double %tmp32, 1.000000e+00
  %tmp34 = and i1 %tmp25, %tmp33
  br i1 %tmp34, label %while_body, label %merge

while_body:                                       ; preds = %while
  %x6 = load double, double* %x
  store double %x6, double* %X
  %y7 = load double, double* %y
  store double %y7, double* %Y
  %X8 = load double, double* %X
  %X9 = load double, double* %X
  %tmp10 = fmul double %X8, %X9
  %Y11 = load double, double* %Y
  %Y12 = load double, double* %Y
  %tmp13 = fmul double %Y11, %Y12
  %tmp14 = fsub double %tmp10, %tmp13
  %tmp15 = fadd double %tmp14, 3.623700e-01
  store double %tmp15, double* %x
  %X16 = load double, double* %X
  %tmp17 = fmul double 2.000000e+00, %X16
  %Y18 = load double, double* %Y
  %tmp19 = fmul double %tmp17, %Y18
  %tmp20 = fadd double %tmp19, 3.200000e-01
  store double %tmp20, double* %y
  %n21 = load double, double* %n
  %n22 = load double, double* %n
  %tmp23 = fadd double %n22, 1.000000e+00
  store double %tmp23, double* %n
  br label %while

merge:                                            ; preds = %while
```

```
  %tmp_log = call double (double, ...) @log(double 2.718281e+00)
  %n35 = load double, double* %n
  %tmp_log36 = call double (double, ...) @log(double %n35)
  %log = fdiv double %tmp_log36, %tmp_log
  %tmp37 = fmul double %log, 1.280000e+02
  %tmp38 = fptosi double %tmp37 to i32
  ret i32 %tmp38
}

define i32 @green(i32 %i, i32 %j) {
entry:
  %i1 = alloca i32
  store i32 %i, i32* %i1
  %j2 = alloca i32
  store i32 %j, i32* %j2
  %x = alloca double
  store double 0.000000e+00, double* %x
  %y = alloca double
  store double 0.000000e+00, double* %y
  %X = alloca double
  store double 0.000000e+00, double* %X
  %Y = alloca double
  store double 0.000000e+00, double* %Y
  %n = alloca double
  store double 0.000000e+00, double* %n
  %i3 = load i32, i32* %i1
  %tmp = call double @dim(i32 %i3)
  store double %tmp, double* %x
  %j4 = load i32, i32* %j2
  %tmp5 = call double @dim(i32 %j4)
  store double %tmp5, double* %y
  br label %while

while:                                            ; preds = %while_body, %entry
  %n24 = load double, double* %n
  %tmp25 = fcmp olt double %n24, 2.000000e+02
  %x26 = load double, double* %x
  %x27 = load double, double* %x
  %tmp28 = fmul double %x26, %x27
  %y29 = load double, double* %y
  %y30 = load double, double* %y
  %tmp31 = fmul double %y29, %y30
  %tmp32 = fadd double %tmp28, %tmp31
  %tmp33 = fcmp olt double %tmp32, 1.000000e+00
  %tmp34 = and i1 %tmp25, %tmp33
  br i1 %tmp34, label %while_body, label %merge

while_body:                                       ; preds = %while
  %x6 = load double, double* %x
  store double %x6, double* %X
  %y7 = load double, double* %y
  store double %y7, double* %Y
  %X8 = load double, double* %X
  %X9 = load double, double* %X
  %tmp10 = fmul double %X8, %X9
  %Y11 = load double, double* %Y
  %Y12 = load double, double* %Y
```

```
%tmp13 = fmul double %Y11, %Y12
  %tmp14 = fsub double %tmp10, %tmp13
  %tmp15 = fsub double %tmp14, 7.000000e-01
  store double %tmp15, double* %x
  %X16 = load double, double* %X
  %tmp17 = fmul double 2.000000e+00, %X16
  %Y18 = load double, double* %Y
  %tmp19 = fmul double %tmp17, %Y18
  %tmp20 = fadd double %tmp19, 2.701500e-01
  store double %tmp20, double* %y
  %n21 = load double, double* %n
  %n22 = load double, double* %n
  %tmp23 = fadd double %n22, 1.000000e+00
  store double %tmp23, double* %n
  br label %while

merge:                                            ; preds = %while
  %tmp_log = call double (double, ...) @log(double 2.718281e+00)
  %n35 = load double, double* %n
  %tmp_log36 = call double (double, ...) @log(double %n35)
  %log = fdiv double %tmp_log36, %tmp_log
  %tmp37 = fmul double %log, 9.600000e+01
  %tmp38 = fptosi double %tmp37 to i32
  ret i32 %tmp38
}

define i32 @red(i32 %i, i32 %j) {
entry:
  %i1 = alloca i32
  store i32 %i, i32* %i1
  %j2 = alloca i32
  store i32 %j, i32* %j2
  %x = alloca double
  store double 0.000000e+00, double* %x
  %y = alloca double
  store double 0.000000e+00, double* %y
  %X = alloca double
  store double 0.000000e+00, double* %X
  %Y = alloca double
  store double 0.000000e+00, double* %Y
  %n = alloca double
  store double 0.000000e+00, double* %n
  %i3 = load i32, i32* %i1
  %tmp = call double @dim(i32 %i3)
  store double %tmp, double* %x
  %j4 = load i32, i32* %j2
  %tmp5 = call double @dim(i32 %j4)
  store double %tmp5, double* %y
  br label %while

while:                                            ; preds = %while_body, %entry
  %n24 = load double, double* %n
  %tmp25 = fcmp olt double %n24, 2.000000e+02
  %x26 = load double, double* %x
  %x27 = load double, double* %x
  %tmp28 = fmul double %x26, %x27
  %y29 = load double, double* %y
```

```
  %y30 = load double, double* %y
  %tmp31 = fmul double %y29, %y30
  %tmp32 = fadd double %tmp28, %tmp31
  %tmp33 = fcmp olt double %tmp32, 1.000000e+00
  %tmp34 = and i1 %tmp25, %tmp33
  br i1 %tmp34, label %while_body, label %merge

while_body:                                       ; preds = %while
  %x6 = load double, double* %x
  %x7 = load double, double* %x
  %tmp8 = fmul double %x6, %x7
  store double %tmp8, double* %X
  %y9 = load double, double* %y
  %y10 = load double, double* %y
  %tmp11 = fmul double %y9, %y10
  store double %tmp11, double* %Y
  %X12 = load double, double* %X
  %Y13 = load double, double* %Y
  %tmp14 = fsub double %X12, %Y13
  %tmp15 = fadd double %tmp14, 3.623700e-01
  store double %tmp15, double* %x
  %x16 = load double, double* %x
  %tmp17 = fmul double 2.000000e+00, %x16
  %y18 = load double, double* %y
  %tmp19 = fmul double %tmp17, %y18
  %tmp20 = fadd double %tmp19, 3.200000e-01
  store double %tmp20, double* %y
  %n21 = load double, double* %n
  %n22 = load double, double* %n
  %tmp23 = fadd double %n22, 1.000000e+00
  store double %tmp23, double* %n
  br label %while

merge:                                            ; preds = %while
  %tmp_log = call double (double, ...) @log(double 2.718281e+00)
  %n35 = load double, double* %n
  %tmp_log36 = call double (double, ...) @log(double %n35)
  %log = fdiv double %tmp_log36, %tmp_log
  %tmp37 = fmul double %log, 2.560000e+02
  %tmp38 = fptosi double %tmp37 to i32
  ret i32 %tmp38
}

define double @dim(i32 %x) {
entry:
  %x1 = alloca i32
  store i32 %x, i32* %x1
  %f = alloca double
  %x2 = load i32, i32* %x1
  %H = load i32, i32* @H
  %tmp = sitofp i32 %H to double
  %tmp3 = fdiv double %tmp, 2.000000e+00
  %tmp4 = sitofp i32 %x2 to double
  %tmp5 = fsub double %tmp4, %tmp3
  %H6 = load i32, i32* @H
  %tmp7 = sitofp i32 %H6 to double
  %tmp8 = fdiv double %tmp7, 2.000000e+00
```

```
  %tmp9 = fdiv double %tmp5, %tmp8
  store double %tmp9, double* %f
  %f10 = load double, double* %f
  ret double %f10
}

define void @graph([960 x i32]* %canvas, i32 %w, i32 %h, i32 (i32, i32)* %paint) {
entry:
  %canvas1 = alloca [960 x i32]*
  store [960 x i32]* %canvas, [960 x i32]** %canvas1
  %w2 = alloca i32
  store i32 %w, i32* %w2
  %h3 = alloca i32
  store i32 %h, i32* %h3
  %paint4 = alloca i32 (i32, i32)*
  store i32 (i32, i32)* %paint, i32 (i32, i32)** %paint4
  %x = alloca i32
  store i32 0, i32* %x
  %y = alloca i32
  store i32 0, i32* %y
  store i32 0, i32* %y
  br label %while

while:                                            ; preds = %merge, %entry
  %y24 = load i32, i32* %y
  %h25 = load i32, i32* %h3
  %tmp26 = icmp slt i32 %y24, %h25
  br i1 %tmp26, label %while_body, label %merge27

while_body:                                       ; preds = %while
  store i32 0, i32* %x
  br label %while5

while5:                                           ; preds = %while_body6, %while_body
  %x18 = load i32, i32* %x
  %w19 = load i32, i32* %w2
  %tmp20 = icmp slt i32 %x18, %w19
  br i1 %tmp20, label %while_body6, label %merge

while_body6:                                      ; preds = %while5
  %canvas7 = load [960 x i32]*, [960 x i32]** %canvas1
  %y8 = load i32, i32* %y
  %canvas9 = getelementptr inbounds [960 x i32], [960 x i32]* %canvas7, i32 %y8
  %x10 = load i32, i32* %x
  %canvas11 = getelementptr inbounds [960 x i32], [960 x i32]* %canvas9, i32 0, i32 %x10
  %paint12 = load i32 (i32, i32)*, i32 (i32, i32)** %paint4
  %y13 = load i32, i32* %y
  %x14 = load i32, i32* %x
  %tmp = call i32 %paint12(i32 %x14, i32 %y13)
  store i32 %tmp, i32* %canvas11
  %x15 = load i32, i32* %x
  %x16 = load i32, i32* %x
  %tmp17 = add i32 %x16, 1
  store i32 %tmp17, i32* %x
  br label %while5

merge:                                            ; preds = %while5
```

```
  %y21 = load i32, i32* %y
  %y22 = load i32, i32* %y
  %tmp23 = add i32 %y22, 1
  store i32 %tmp23, i32* %y
  br label %while

merge27:                                          ; preds = %while
  ret void
}

declare i32 @draw_default(...)

declare i32 @do_draw(i32*, i32, i32, i32, i32, ...)

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

declare double @pow(double, double)

declare double @sin(double, ...)

declare double @cos(double, ...)

declare double @tan(double, ...)

declare double @log(double, ...)

declare double @rando()

declare double @randos(i32)

define i32 @main() {
entry:
  store i32 960, i32* @W
  store i32 720, i32* @H
  %H = load i32, i32* @H
  %W = load i32, i32* @W
  call void @graph([960 x i32]* getelementptr inbounds ([720 x [960 x i32]], [720 x [960 x
i32]]* @canv, i32 0, i32 0), i32 %W, i32 %H, i32 (i32, i32)* @paint_jset)
  %do_draw = call i32 (i32*, i32, i32, i32, i32, ...) @do_draw(i32* getelementptr inbounds
([720 x [960 x i32]], [720 x [960 x i32]]* @canv, i32 0, i32 0, i32 0), i32 960, i32 720, i32
0, i32 0)
  ret i32 0
}
```

Figure 4: jset LLVM IR code

6.2 Test Suite

We created our test suite to help both with ensuring that our scanner and parser were functioning correctly and also to check that we were getting the expected output once we began generating LLVM IR code. The test cases are stored in the folder called “tests”. The test cases include the easel programs as well as their respective AST outputs and either the compiler normal output or the failure outputs, depending on the type of test. Most of our tests cases aim to cover different aspects of the easel language, including everything relating to statements and functions covered in our language reference manual.

Test cases were written when new features were added to the language. At first the AST test cases was created when the scanner and parser were created to ensure that there was no ambiguity in the grammar. Then fail cases for semantic testing were created to be used to confirm that our semantic checker was functioning as planned. As we wrote codegen, the test cases for the expected output of programs were added to test that we were correctly compiling down to LLVM IR. For the purposes of testing output, we created a few print statements; however, easily normally would not allow a user to print to the screen because the program is meant for drawing images. We additionally created test programs to ensure that drawing images was working as expected.

We performed the following types of testing:
1. Literals (declaration, comparison, assignment, type casting)
2. Functions (nested, user-defined, built-in, recursive calls, functions passed as arguments, anonymous)
3. Scope (global, local)
4. Statements (if, while, for, return, nested statements)
5. Array (assignment, arrays are function arguments)
6. Pix (declaration, operations, matrix operations)
7. Draw

All files in our test suite are listed in the Appendix. We included tests that were expected to pass, as well as tests that were expected to fail. We also created tests that draw easel graphs (also those were checked by visual confirmation).

6.3 Test Automation

In order to easily run test cases in our language, we created a shell script (autotest.sh) that will automatically run every test in our tests/ directory. This test script will diff the outputs of the tests with the expected outputs that we created. If the test passes, it will print "test-xxx…OK", otherwise it will return "fail-xxx.es...FAILED" and then print out the failure information such as "fail-xx.err differs". If the test does not pass, the respective diff files and expected test output for the failed tests will be generated and remain in the easel/ repository for debugging. And we can also use autotest.sh -k argument to keep the intermediate generated files (such as expected failure .err files, expected normal output .out files) in the repository for finding and resolving bugs. To test our scanner and parser, we used the –a option to return the AST output.

```
plt4115@plt4115:~/easel$ ./autotest.sh
make: Nothing to be done for 'all'.
ocamlbuild -use-ocamlfind -pkgs llvm,llvm.analysis -cflags -w,+a-4 \
        easel.native
Finished, 22 targets (22 cached) in 00:00:00.
test-anonfunc1...OK
test-anonfunc2...OK
test-arith2...OK
test-arith-difftype1...OK
test-arith-difftype2...OK
test-arith...OK
test-arr-args1...OK
test-arr-args2...OK
test-array1...OK
test-array2...OK
test-array3...OK
test-cirfun...OK
```

6.4 Testing Roles

The majority of the test cases were written by Sophie. She looked both through the LRM and at other code that was being pushed (and that she wrote herself) and made tests based on those. Oswin, Danielle, and Sophie contributed to the code in the demos/ directory that is used to test that the draw function works. All team members contributed a few test cases to both the test and fail cases. The whole team also looked over the test suite to ensure that all cases were being covered and we recommended cases that should be written if they did not yet exist.

7. Lessons Learned

7.1 Danielle Crosswell

This semester is easily the longest time I have spent working on a project for a class. What made this project (almost) completely stress-free was my team. If I learned one thing from this semester of group projects in almost all of my classes, it is that you need a good team. I learned from this assignment that there are in fact people at this school that I can depend on to complete a project by deadlines and pick up my slack if I'm having a rough week; however, with a great team comes great responsibility.

It is really easy to get swamped with other work and take a week off from working on this project. Your language is constantly changing and evolving, whether you're moving with it or not. I learned that if you stop keeping up with the new commits and functions being added to your language, you will get lost. It's really easy to fall behind in something like this and the most important thing I learned is to at least keep up with the code, even if you don't have time to contribute anything meaningful.

Another problem I ran into was not completely understanding OCaml in the beginning. We were fortunate because the base of our language is similar to MicroC. We were able to get started pretty easily and had a nice code repository to reference when we got stuck; however, our code very quickly diverged from MicroC and I got stuck trying to force MicroC-esque code to work on easel. Rather than actually understanding the code and easel itself, I wasted a lot of time in the beginning trying to replicate MicroC. From this experience, I learned to understand first and code second. This project could be really challenging and frustrating at times but being able to finally create the image we had set as our goal (the mandelbrot) was incredibly rewarding.

7.2 Tyrus Cukavac

As the language guru, I think the most important lesson I learned was not to get too wrapped up in details while defining the language. Obviously, a well-defined language requires specific and unambiguous details. However, in my effort to do too much too quickly while creating the language reference manual, I found myself getting stuck in the weeds and into paradoxical thought loops that made the creation of the LRM incredibly overwrought and confusing.

In hindsight, I wish I had stuck with principles of our language while creating the initial draft of the LRM (almost beginning with a language tutorial and then deriving the grammar from there) rather than trying to force a grammar and attempting (poorly) to create a readable document. Lesson learned: don't get stuck in the weeds and abide by your principles and goals as much as possible!

On the subject of principles and goals, I think I started with a number of irreconcilable and unnecessary objectives rather than looking more holistically at the function that easel could serve and how best it could achieve this. There was too much hand-wringing on my part early on along

the lines of "Well, it needs to be pseudo-functional!" and "We need to have tuples!" and "It needs to divide by zero!" (Kidding on the last sentence, although during implementation it definitely often felt like I was asking for impossible things!) Instead, a more attainable goal might have been "a simple language that uses functions to make pictures". With such a clean objective, useful features could be derived in a cleaner, less harried, and more organic manner. With the earlier approach, at times it felt like I was designing the Homer Simpson car.

Another important lesson - good teammates makes for a good project. I feel very fortunate to have worked with such a supportive and talented group of people that made the creation of our language possible.

7.3 Oswin Chou

Quite an enjoyable pain! I mean, OCaml- the functional programming language. Do not expect that you can pick up this language as quick as you pick up others. Coding in a functional language is completely a different experience from what I do in other prevailing languages like Python, JavaScript, C, etc. Sometimes it can get people stuck for hours just for a few lines of code. So just like what ancient warriors do for preparing for battles, sharpen your weapon before it starts! I wish I had read through the code of MicroC earlier so I could come up with solutions quicker and make better choices when writing the code. However, everything costs. Doing additional reading work does the same- it costs you time. Hence, for me I think it would be better to have only this one heavy project-implementing course, so I can devote myself to this fulfilling project. But it is worth doing. I might have no other chances in my life to code in a functional language so seriously. And it is always interesting to learn and play with something different, so you can have fun and have a broader horizon.

And the other thing about what everyone suggests, making a good choice on teammates, I think what you need is luck- you cannot know how good they are after you really work with them. I am fortunate to have enough luck to team up with these amazing people- so talented, resourceful, and energetic. Everyone does their job so well and always helps others without hesitance. That is what you really need to do a project well!

7.4 Sophie Chen

As the tester, the first thing I learned from this project is that the test suite needs to be built as soon as possible. This great suggestion is made by Rachel on the first day of our meetings. And working through your LRM is a good idea to create the test suite. Although some of the test cases may not pass before the complete implementation of the language, it could be a guide for you to see how much work is left to be done.

During the implementation of the language, it is really easy to replicate others' codes that seems to but actually does not fit in your language, so please don't do that. You should first refer to the commits in the repo to catch up with the changes of your language, and communicate timely with your teammates. Otherwise it will make unnecessary conflicts in the codes. Thanks to the constant communications between our team, this kind of situation happens rarely.

And in the semester-long project, the most important thing I have learned is teamwork. I am very lucky to work with such well-rounded and talented teammates. Everyone in our team is willing to find out solutions to the unsolved problems and never hesitate to give a helping hand to someone who was stuck in any kind of problem. And we also met every week and had "hackathons" together, when my teammates always turned the painful coding process of OCaml

into efficient learning time along with many laughs. Coding with my teammates is definitely one of the most enjoyable times in Columbia!

7.5 Advice for Future Teams

Our first piece of advice for future teams is to choose a project that you are all excited about. Think more about the type of work you would like to accomplish rather than what you think you are able to accomplish. We also highly recommend that you use git - not only is it required to submit your project log, but it also keeps everything extremely organized and allows you to revert to a previous version when you mess something up horribly. When using git, have one person (your system architect) in charge of making your repository and managing pull requests. That way one person has the final say on the code being merged into your project.

Our next piece of advice is to create small goals for yourself. Once we began code generation, we started by implementing the features needed for Hello World to work. We then began working towards making the mandelbrot run. Once those two goals were met, our final step was to complete our language. However, something important to note here is that even if you think your code is almost done, you're probably wrong. Just because you hit a milestone doesn't mean that you can stop working. This project will constantly have work to be done until you submit your final code directory. Along this line, try to work chronologically through your code; don't put off other programs, such as your semantic checker, just because you need to start working on codegen. Try to stay on top of your work and get one part of the pipeline done before moving onto the next. Finally, work together and be communicative. Clearly set up who in your team is expected to accomplish certain goals and let your team know when you are working through those assignments. You don't want to run into the problem where two people are implementing the same piece of code simultaneously. More useful than being communicative though is finding a few hours during the weekend to all sit together and code. It is much easier to work through code when you have resources that you can get help from when you get stuck on a problem, rather than sitting alone in your room for five hours trying to figure it out yourself (and failing).

8. Appendix

8.1 scanner.mll

```
(* scanner.mll *)
(* By: Oswin, Tyrus *)
(* Ocamllex scanner for easel *)

{ open Parser }

let digit = ['0'-'9']
let hexdigit = ['0'-'9' 'a'-'f' 'A'-'F']
let exp = 'e' ['+' '-']? digit+

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"     { comment 1 lexbuf }           (* Comments *)
| '('      { LPAREN }
| ')'      { RPAREN }
| '['      { LBRCK }
| ']'      { RBRCK }
| '{'      { LBRACE }
| '}'      { RBRACE }
```

```
| ';'      { SEMI }
| ','      { COMMA }
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '%'      { MOD }
| '^'      { POW }
| '.'      { DOT }
| "++"     { INC }
| "--"     { DEC }
| '='      { ASSIGN }
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
| ">"      { GT }
| ">="     { GEQ }
| "&&"     { AND }
| "||"     { OR }
| "!"      { NOT }
| "if"     { IF }
| "else"   { ELSE }
| "for"    { FOR }
| "while"  { WHILE }
| "return" { RETURN }
| "int"    { INT }
| "float"  { FLOAT }
| "bool"   { BOOL }
| "void"   { VOID }
| "pix"    { PIX }
| "function"  { FUNC }
| "true" | "false" as lxm { BOOLLIT(bool_of_string lxm) }
(* No harm to support 0xFF format *)
| '0'('x'|'X') hexdigit+ | digit+ as lxm { INTLIT(int_of_string lxm) }
| '#' hexdigit+ as lxm {
    let lit = "0x" ^ String.sub lxm 1 (String.length lxm - 1) in
    INTLIT(int_of_string lit)
  }
(* Float literal *)
| (digit* '.' digit+ | digit+ '.') exp? | digit+ exp as lxm
  { FLOATLIT(float_of_string lxm) }
(* Identifier *)
| ['a'-'z' 'A'-'Z' '_']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

and comment cnt = parse
  "/*" { comment (cnt + 1) lexbuf }
|  "*/" { if cnt = 1 then token lexbuf else comment (cnt - 1) lexbuf }
| _    { comment cnt lexbuf }
```

8.2 parser.mly

```
/* parser.mly */
/* By: Oswin, Tyrus, Danielle */
/* Ocamlyacc parser for easel */

%{
open Ast
%}

%token FUNC
%token SEMI LPAREN RPAREN LBRCK RBRCK LBRACE RBRACE COMMA
%token PLUS MINUS TIMES DIVIDE MOD POW ASSIGN NOT
%token INC DEC DOT
%token EQ NEQ LT LEQ GT GEQ AND OR
%token RETURN IF ELSE FOR WHILE INT FLOAT BOOL VOID PIX
%token <int> INTLIT
%token <float> FLOATLIT
%token <bool> BOOLLIT
%token <string> ID
%token EOF

%nonassoc NOELSE
%nonassoc ELSE

%start program
%type <Ast.program> program

%%

program:
  decls EOF { $1 }

decls:
    /* nothing */ { [], [] }
 | decls fdecl { let (fds, sts) = $1 in ($2 :: fds), sts }
 | decls stmt { let (fds, sts) = $1 in fds, ($2 :: sts) }

fdecl:
    FUNC typ ID LPAREN formals RPAREN LBRACE stmt_list RBRACE
     { { typ = $2;
        fname = $3;
        formals = List.rev $5;
     body = List.rev $8;
     checked = false } }
  | FUNC typ ID LPAREN RPAREN LBRACE stmt_list RBRACE
     { { typ = $2;
        fname = $3;
     formals = [];
        body = List.rev $7;
     checked = false } }
  | FUNC typ ID LPAREN formals RPAREN LBRACE RBRACE
     { { typ = $2;
        fname = $3;
        formals = List.rev $5;
     body = [];
     checked = false } }
  | FUNC typ ID LPAREN RPAREN LBRACE RBRACE
```

```
     { { typ = $2;
         fname = $3;
     formals = [];
     body = [];
     checked = false } }

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

aformals:
    typ                     { [$1] }
  | aformals COMMA typ { $3 :: $1 }

typ:
    prim_typ { $1 }
  | afunc_typ { $1 }
  | typ LBRCK RBRCK { ArrRef($1, 0) }
  | typ LBRCK INTLIT RBRCK { ArrRef($1, $3) }

prim_typ:
    INT { Int }
  | FLOAT { Float }
  | BOOL { Bool }
  | VOID { Void }
  | PIX { Pix }

afunc_typ:
    FUNC typ LPAREN aformals RPAREN
    { Func($2, List.rev $4) }
  | FUNC typ LPAREN RPAREN
    { Func($2, []) }

init_dectr_list:
    init_dectr    { [$1] }
  | init_dectr_list COMMA init_dectr { $3 :: $1 }

init_dectr:
    dectr { InitDectr($1, Noexpr) }
  | dectr ASSIGN expr { InitDectr($1, $3) }

dectr:
    ID { DecId($1) }
  | dectr LBRCK INTLIT RBRCK { DecArr($1, $3) }

stmt_list:
    stmt           { [$1] }
  | stmt_list stmt { $2 :: $1 }

stmt:
    expr SEMI { Expr $1 }
  | typ init_dectr_list SEMI { Vdef($1, $2) }
  | RETURN SEMI { Return Noexpr }
  | RETURN expr SEMI { Return $2 }
  | LBRACE stmt_list RBRACE { Block(List.rev $2) }
  | LBRACE RBRACE { Block([]) }
  | 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) }

anonfunc:
    FUNC typ LPAREN formals RPAREN LBRACE stmt_list RBRACE
    { AnonFunc({ typ = $2; fname = ""; formals = List.rev $4; body = List.rev $7; checked =
false }) }
  | FUNC typ LPAREN RPAREN LBRACE stmt_list RBRACE
    { AnonFunc({ typ = $2; fname = ""; formals = []; body = List.rev $6; checked = false }) }
  | FUNC typ LPAREN formals RPAREN LBRACE RBRACE
    { AnonFunc({ typ = $2; fname = ""; formals = List.rev $4; body = []; checked = false }) }
  | FUNC typ LPAREN RPAREN LBRACE RBRACE
    { AnonFunc({ typ = $2; fname = ""; formals = []; body = []; checked = false }) }

expr_opt:
    /* nothing */ { Noexpr }
  | expr          { $1 }

expr:
    assign_expr { $1 }

assign_expr:
    logic_or_expr                   { $1 }
  | anonfunc                        { $1 }
  | LBRACE expr COMMA expr COMMA expr COMMA expr RBRACE { PixLit($2, $4, $6, $8) }
  | postfix_expr ASSIGN assign_expr { Assign($1, $3) }

logic_or_expr:
    logic_and_expr { $1 }
  | logic_or_expr OR logic_and_expr { Binop($1, Or, $3) }

logic_and_expr:
    eq_expr { $1 }
  | logic_and_expr AND eq_expr { Binop($1, And, $3) }

eq_expr:
    rel_expr { $1 }
  | eq_expr EQ rel_expr { Binop($1, Equal, $3) }
  | eq_expr NEQ rel_expr { Binop($1, Neq, $3) }

rel_expr:
    add_expr { $1 }
  | rel_expr LT add_expr { Binop($1, Less, $3) }
  | rel_expr GT add_expr { Binop($1, Greater, $3) }
  | rel_expr LEQ add_expr { Binop($1, Leq, $3) }
  | rel_expr GEQ add_expr { Binop($1, Geq, $3) }

add_expr:
    mult_expr { $1 }
  | add_expr PLUS mult_expr { Binop($1, Add, $3) }
  | add_expr MINUS mult_expr { Binop($1, Sub, $3) }

mult_expr:
    exp_expr { $1 }
  | mult_expr TIMES exp_expr { Binop($1, Mult, $3) }
```

```
  | mult_expr DIVIDE exp_expr { Binop($1, Div, $3) }
  | mult_expr MOD exp_expr { Binop($1, Mod, $3) }

exp_expr:
    unary_expr { $1 }
  | exp_expr POW unary_expr { Binop($1, Pow, $3) }

unary_expr:
    postfix_expr { $1 }
  | PLUS unary_expr { $2 }
  | MINUS unary_expr { Unop(Neg, $2) }
  | NOT unary_expr { Unop(Not, $2) }

postfix_expr:
    base_expr { $1 }
  | postfix_expr LBRCK expr RBRCK { EleAt($1, $3) }
  | postfix_expr LPAREN actuals_list RPAREN { Call($1, List.rev $3) }
  | postfix_expr LPAREN RPAREN { Call($1, []) }
  | postfix_expr DOT ID { PropAcc($1, $3) }
  | postfix_expr INC { Unop(Inc, $1) }
  | postfix_expr DEC { Unop(Dec, $1) }

base_expr:
    INTLIT             { IntLit($1) }
  | FLOATLIT           { FloatLit($1) }
  | BOOLLIT            { BoolLit($1) }
  | ID                 { Id($1) }
  | LPAREN expr RPAREN { $2 }

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

8.3 ast.ml

```
(* ast.ml *)
(* By: Oswin, Tyrus, Danielle and Sophie *)
(* Abstract Syntax Tree and functions for printing it *)

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

and uop = Neg | Not | Inc | Dec

and typ = Int | Float | Bool | Void | Pix | Func of typ * typ list | ArrRef of typ * int

and dectr =
    DecId of string
  | DecArr of dectr * int (* declarator * array length *)

and bind = typ * dectr
```

```
and expr =
    IntLit of int
  | FloatLit of float
  | BoolLit of bool
  | PixLit of expr * expr * expr * expr
  | Id of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Assign of expr * expr
  | Call of expr * expr list
  | EleAt of expr * expr
  | PropAcc of expr * string (* Access property "string" of "expr" *)
  | AnonFunc of func_decl
  | Noexpr

and init_dectr = InitDectr of dectr * expr (* dectr * Initializer *)

and stmt =
    Block of stmt list
  | Expr of expr
  | Vdef of typ * init_dectr list
  | Return of expr
  | If of expr * stmt * stmt
  | For of expr * expr * expr * stmt
  | While of expr * stmt

and func_decl = {
    typ : typ;
    fname : string;
    formals : bind list;
    body : stmt list;
    checked: bool;
  }

and program = func_decl list * stmt list

(* Pretty-printing functions *)

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

and string_of_typ = function
    Int -> "int"
  | Float -> "float"
```

```
  | Bool -> "bool"
  | Void -> "void"
  | Pix -> "pix"
  | Func(t, tl) ->
    "function " ^ string_of_typ t ^ " (" ^ String.concat ", " (List.map string_of_typ tl) ^
")"
  | ArrRef(t, 0) -> string_of_typ t ^ "[]"
  | ArrRef(t, l) -> string_of_typ t ^ "[" ^ string_of_int l ^ "]"

and string_of_dectr = function
    DecId(s) -> s
  | DecArr(d, 0) -> string_of_dectr d ^ "[]"
  | DecArr(d, l) -> string_of_dectr d ^ "[" ^ string_of_int l ^ "]"

and string_of_bind (t, d) =
    string_of_typ t ^ " " ^ string_of_dectr d

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

and string_of_expr = function
    IntLit(l) -> string_of_int l
  | FloatLit(f) -> string_of_float f
  | BoolLit(b) -> string_of_bool b
  (*| ArrLit(el) -> "[" ^ String.concat ", " (List.map string_of_expr el) ^ "]"*)
  | PixLit(e1, e2, e3, e4) -> "{" ^ string_of_expr e1 ^ ", " ^ string_of_expr e2 ^ ", " ^
string_of_expr e3 ^  ", " ^ string_of_expr e4 ^ "}"
  | Id(s) -> s
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
  | Unop(o, e) ->
      match o with
      Neg -> "-" ^ string_of_expr e
    | Not -> "!" ^ string_of_expr e
    | Inc -> string_of_expr e ^ "++"
    | Dec -> string_of_expr e ^ "--";
    ;
  | Assign(v, e) -> string_of_expr v ^ " = " ^ string_of_expr e
  | Call(f, el) ->
      string_of_expr f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | EleAt(arr, idx) -> string_of_expr arr ^ "[" ^ string_of_expr idx ^ "]"
  | PropAcc(e, id) -> string_of_expr e ^ "." ^ id
  | AnonFunc(func) -> string_of_fdecl func
  | Noexpr -> ""

and string_of_initdectr = function
    InitDectr(s, Noexpr) -> string_of_dectr s
  | InitDectr(s, e) -> string_of_dectr s ^ " = " ^ string_of_expr e

and string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | Vdef(t, dl) -> string_of_typ t ^ " " ^
```

```
      String.concat ", " (List.map string_of_initdectr (List.rev dl)) ^ ";\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

and string_of_fdecl fdecl =
  "function " ^ string_of_typ fdecl.typ ^ " " ^
  fdecl.fname ^ "(" ^ String.concat ", " (List.map string_of_bind fdecl.formals) ^
  ")\n{\n" ^
  String.concat "" (List.map string_of_stmt fdecl.body) ^
  "}"

and string_of_funcs = function
    [] -> ""
  | funcs -> String.concat "\n\n" (List.map string_of_fdecl (List.rev funcs)) ^ "\n\n"

and string_of_program (funcs, stmts) =
  string_of_funcs funcs ^
  String.concat "" (List.map string_of_stmt (List.rev stmts))
```

8.4 semant.ml

```
(* semant.ml *)
(* By: Oswin, Tyrus, Sophie *)
(* Semantic checking for easel compiler *)

open Ast

module StringMap = Map.Make(String)

(* The semantic checker will return void if successful and will throw an exception otherwise
*)

let globals = Hashtbl.create 8;;

let check (functions, statements) =

  (* check for duplicates names *)
  let report_dup exceptf list =
      let rec helper = function
          n1 :: n2 :: _ when n1 = n2 -> raise (Failure (exceptf n1))
        | _ :: t -> helper t
        | [] -> ()
      in helper (List.sort compare list)
  in

  (* check for void type *)
  let check_void exceptf = function
        (Void, n) -> raise (Failure (exceptf n))
      | _ -> ()
  in
```

```
  (* check that rvalue type can be assigned to lvalue type *)
  let check_assign lvalt rvalt err = match (lvalt, rvalt) with
        (Pix, Int) | (Int, Pix) | (ArrRef(Pix, _), ArrRef(Int, _)) |
        (ArrRef(ArrRef(Pix, _), _), ArrRef(ArrRef(Int, _), _)) -> lvalt
      | (ArrRef(ArrRef(lv,_),_), ArrRef(ArrRef(rv,_),_)) -> if lv = rv then lvalt else raise
err
      | (lv, rv) -> if lv = rv then lvalt else raise err
  in

  if List.mem "print" (List.map (fun fd -> fd.fname) functions)
  then raise (Failure ("function print may not be defined")) else ();

  (* Check and build function table *)
  let functions =
        { typ = Void; fname = "draw_default"; formals = [];
          body = []; checked = true }::
        { typ = Void; fname = "do_draw"; formals = [(Pix, DecArr(DecArr(DecId("canvas"), 0),
0));
              (Int, DecId("w")); (Int, DecId("h")); (Int, DecId("x")); (Int, DecId("y"))];
          body = []; checked = true }::
        { typ = Void; fname = "draw"; formals = [(Pix, DecArr(DecArr(DecId("canvas"), 0), 0));
(Int, DecId("x")); (Int, DecId("y"))];
          body = []; checked = true }::
        { typ = Void; fname = "draw_size"; formals = [(Pix, DecArr(DecArr(DecId("canvas"), 0),
0));
               (Int, DecId("w")); (Int, DecId("h")); (Int, DecId("x")); (Int, DecId("y"))];
          body = []; checked = true }::
        { typ = Void; fname = "print"; formals =  [(Int, DecId("x"))];
          body = []; checked = true }::
        { typ = Void; fname = "printfl"; formals =  [(Float,  DecId("x"))];
          body = []; checked = true }::
        { typ = Void; fname = "printp"; formals =  [(Pix,  DecId("x"))];
          body = []; checked = true }::
        { typ = Void; fname = "printb"; formals =  [(Bool,  DecId("x"))];
          body = []; checked = true }::

        { typ = Float; fname = "pow"; formals = [(Float, DecId("x")); (Float, DecId("y"))];
          body = []; checked = true }::
        { typ = Float; fname = "tan"; formals = [(Float, DecId("x"))];
          body = []; checked = true }::
        { typ = Float; fname = "sin"; formals = [(Float, DecId("x"))];
          body = []; checked = true }::
        { typ = Float; fname = "cos"; formals = [(Float, DecId("x"))];
          body = []; checked = true }::
        { typ = Float; fname = "log"; formals= [(Float, DecId("base"));
(Float,  DecId("value"))];
          body = []; checked = true }::
        { typ = Float; fname = "rando"; formals = [];
          body = []; checked = true }::
        { typ = Float; fname = "randos"; formals = [(Int, DecId("seed"))];
          body = []; checked = true }:: functions
  in

    let rec typ_of_bind = function
          (ArrRef(ArrRef(t, _),_),DecId(_)) -> ArrRef(ArrRef(t, 0),0)
        | (ArrRef(t, _),DecId(_)) -> ArrRef(t, 0)
        | (t, DecArr(d, _)) -> typ_of_bind (ArrRef(t, 0), d)
```

```
        | (t, DecId(_)) -> t
    in

    let func_sign fd =
        fd.fname
    in

  report_dup (fun n -> "duplicate function " ^ n)
      (List.map (fun fd -> func_sign fd) functions);

  let func_decls = Hashtbl.create 8 in
  let _ =  List.iter (fun fd -> let formal_types =
                List.map (fun formals -> fst formals) fd.formals in
                ignore(Hashtbl.add globals fd.fname (Func(fd.typ, formal_types)));
                Hashtbl.add func_decls (func_sign fd) fd) functions
  in

  let func_checked fd =
      Hashtbl.remove func_decls (func_sign fd);
      Hashtbl.add func_decls (func_sign fd) { fd with checked = true }
  in

  let func_decl func_locals s = try StringMap.find s func_locals
      with Not_found ->
          try Hashtbl.find func_decls s
            with Not_found -> raise (Failure ("unrecognized function " ^ s))
  in

    let type_of_identifier locals id =
      try StringMap.find id locals
        with Not_found ->
          try Hashtbl.find globals id
            with Not_found -> raise (Failure ("undeclared identifier " ^ id))
    in

    let rec id_of_dectr = function
        DecArr(DecArr(DecArr(_, _),_),_) -> raise(Failure("Matrices of greater than 2
dimensions are not supported in easel."))
      | DecArr(d, _) -> id_of_dectr d
      | DecId(id) -> id
    in

    (* Return the type of an expression or throw an exception *)
    let rec expr locals func_locals = function
        IntLit _ -> Int
      | FloatLit _ -> Float
      | BoolLit _ -> Bool
      | PixLit(er, eg, eb, ea)-> (*match el with [e1; e2; e3] -> *)
        let tr = expr locals func_locals er and tg = expr locals func_locals eg and
             tb = expr locals func_locals eb and ta = expr locals func_locals ea in
        if (tr = Int && tg = Int && tb = Int && ta = Int) then Pix
        else raise(Failure ("illegal pix value [" ^ string_of_expr er ^ string_of_expr eg ^
string_of_expr eb ^ string_of_expr ea ^ "]"))
      | Id s -> type_of_identifier locals s
      | Binop(e1, op, e2) as e -> let t1 = expr locals func_locals e1 and t2 = expr locals
func_locals e2 in
        let t = (match (t1,t2) with
```

```
                    (Int,Int) | (Int, Pix) | (Pix, Int) -> Int
                  | (Pix, Pix) -> Pix
                  | (Float,Float) | (Int,Float) | (Float,Int) | (Float, Pix) | (Pix, Float) ->
Float
                  | (Bool, Bool) -> Bool
                  | (_,_) -> Void) in
          (match op with
            Add | Sub | Mult | Div | Mod -> (match t with
                                               Int -> Int
                                             | Pix -> Pix
                                             | Float -> Float
                                             | _ -> raise(Failure("illegal binary operator " ^
                                                    string_of_typ t1 ^ " " ^ string_of_op op
^ " " ^
                                                    string_of_typ t2 ^ " in " ^
string_of_expr e)))
          | Pow when (t = Int || t = Float) -> Float
          | Equal | Neq when (t = Int || t = Pix || t=Float) -> Bool
          | Less | Leq | Greater | Geq when (t = Int || t=Pix || t=Float) -> Bool
          | And | Or when t = Bool -> Bool
          | _ -> raise (Failure ("illegal binary operator " ^
            string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
            string_of_typ t2 ^ " in " ^ string_of_expr e))
          )
      | Unop(op, e) as ex -> let t = expr locals func_locals e in
        (match op with
            Neg -> (match t with
                       Int -> Int
                     | Float -> Float
                     | _ -> raise(Failure("Illegal use of " ^ string_of_uop op ^ " with " ^
string_of_typ t)))
          | Not when t = Bool -> Bool
          | Inc | Dec -> (match t with
                            Int -> Int
                          | Float -> Float
                          | Pix -> Pix
                          | _ -> raise (Failure ("illegal unary operator " ^ string_of_uop op ^
                                                 string_of_typ t ^ " in " ^ string_of_expr ex)))
          | _ -> raise (Failure ("illegal unary operator " ^ string_of_uop op ^
           string_of_typ t ^ " in " ^ string_of_expr ex))
        )
      | Noexpr -> Void
      | Assign(var, e) as ex ->
          let lt = expr locals func_locals var
          and rt = expr locals func_locals e in
            check_assign lt rt (Failure ("illegal assignment " ^
            string_of_typ lt ^ " = " ^ string_of_typ rt ^ " in " ^ string_of_expr ex))
      | Call(fdectr, actuals) as call ->
         (match fdectr with
           Id fname -> let fsign = fname in
         let fd = func_decl func_locals fsign in
         if List.length actuals != List.length fd.formals then
           raise (Failure ("expecting " ^ string_of_int
             (List.length fd.formals) ^ " arguments in " ^ string_of_expr call))
         else
           List.iter2 (fun b e -> let bt = typ_of_bind b in let et = expr locals func_locals e
in
```

```
              ignore (check_assign bt et
                (Failure ("illegal actual argument found " ^ string_of_typ et ^
                " expected " ^ string_of_typ bt ^ " in " ^ string_of_expr e))))
             fd.formals actuals;
           if not fd.checked then (func_checked fd; check_func fd) else ();
           fd.typ
         | _ -> raise(Failure(string_of_expr fdectr ^ " is not a valid function to call" )))
      | EleAt(arr, idx) as ele-> let idxt = expr locals func_locals idx in
                                 if idxt != Int then raise(Failure(string_of_expr idx ^ " of
type " ^ string_of_typ idxt ^
                                   " is not a valid array index for " ^ string_of_expr arr))
                                 else (match arr with
                                        EleAt(iarr, _) -> let iat = expr locals func_locals
iarr in
                                          (match iat with
                                            ArrRef(ArrRef(arr_t, _), _) -> arr_t
                                          | _ -> raise(Failure (string_of_expr ele ^ " is not
a valid array")))
                                      | _ -> let iat = expr locals func_locals arr in
                                             (match iat with
                                               ArrRef(ArrRef(arr_t, _), _) -> ArrRef(arr_t, 0)
                                             | ArrRef(arr_t, _) -> arr_t
                                             | _ -> raise(Failure (string_of_expr ele ^ " is
not a valid array"))))
      | PropAcc(e, prp) ->
          (* Find the type of a given thing *)
          let t = expr locals func_locals e in
          (* Make sure the property works for the type *)
          (match t with
              Pix -> (match prp with
                          "red" | "green" | "blue" | "alpha"-> Int
                        | _ -> raise(Failure ("invalid pixel property " ^ prp)))
            | ArrRef(_, _) -> (match prp with
                                "size" -> Int
                              | _ -> raise(Failure ("invalid array property " ^ prp)))
            | _ -> raise(Failure ("type " ^ string_of_typ t ^ "has no valid property " ^
prp)))
      | AnonFunc(func_decl) -> let formal_types = List.map (fun (ftyp, _) -> ftyp)
func_decl.formals  in
                                           Func(func_decl.typ, formal_types)

          and check_func func = report_dup (fun _ -> "Duplicate formals in function " ^
func.fname) func.formals;
            List.iter (check_void (fun n -> "Formal arguments cannot have a void type" ^
string_of_dectr n)) func.formals;
          let func_formals = List.fold_left (fun m (typ, dect) -> (match typ with
                                                                    Func (t,f) -> let
form_func_sign = (string_of_dectr dect) in
                                                                                  let
form_form_bind =
                                                                                    List.map
(fun fo -> (fo, DecId("novar"))) f in
                                                                                  let fd =
{typ = t; fname = string_of_dectr dect;
                                                                                    formals =
form_form_bind;
```

```
form_form_bind;
                                                                                body=[];
checked=true} in
                                                                                  StringM
ap.add form_func_sign fd m
                                                                  | _ -> m)) StringMap.empty
func.formals in
          let formals = List.fold_left (fun m (typ, dect) -> StringMap.add (string_of_dectr
dect) typ m) StringMap.empty func.formals in
            check_stmt formals func_formals func.typ (Block func.body)

          and check_vdef l fl t = function
            InitDectr(d, Noexpr) -> typ_of_bind (t, d)
          | InitDectr(d, e) as initd ->
              let lt = typ_of_bind (t, d)
              and rt = expr l fl e in
                check_assign lt rt (Failure ("illegal initialization " ^ string_of_typ lt ^
                  " = " ^ string_of_typ rt ^ " in " ^ string_of_typ t ^ " " ^
string_of_initdectr initd))

          and add_locals locals func_locals t initds =
            List.fold_left (fun m initd -> match initd with InitDectr(d, _) ->
              let tt = check_vdef m func_locals t initd in
              let id = id_of_dectr d in
              if not (StringMap.mem id m) then StringMap.add id tt m
              else raise (Failure ("duplicate local " ^ id))
            ) locals initds

          and check_block locals func_locals funct = function
            [Return _ as s] -> check_stmt locals func_locals funct s
          | Return _ :: _ -> raise (Failure "nothing may follow a return")
          | Block sl :: ss -> check_block locals func_locals funct sl; check_block locals
func_locals funct ss
          | Vdef(t, initds) :: ss -> check_block (add_locals locals func_locals t (List.rev
initds)) func_locals funct ss
          | s :: ss -> check_stmt locals func_locals funct s; check_block locals func_locals
funct ss
          | [] -> ()

          and check_bool_expr l fl e = if expr l fl e != Bool
            then raise (Failure ("expected Boolean expression in " ^ string_of_expr e))
            else ()

          and check_return rt funct = match (rt, funct) with
                                        (Float, Pix) | (Pix, Float) | (Pix, Int) | (Int,
Pix) | (Float, Int) | (Int, Float) -> true
                                      | (_,_) when rt=funct -> true
                                      | (_,_) -> false

          and check_stmt locals func_locals funct = function
            Block sl -> check_block locals func_locals funct sl
          | Expr e -> ignore (expr locals func_locals e)
          | Return e -> let t = expr locals func_locals e in if (check_return t funct) then ()
else
              raise (Failure ("return gives " ^ string_of_typ t ^ " expected " ^
                        string_of_typ funct ^ " in " ^ string_of_expr e))
```

```
                                check_stmt locals func_locals funct b2
          | For(e1, e2, e3, st) -> ignore (expr locals func_locals e1); check_bool_expr locals
func_locals e2;
                                   ignore (expr locals func_locals e3); check_stmt locals
func_locals funct st
          | While(p, s) -> check_bool_expr locals func_locals p; check_stmt locals func_locals
funct s
          | Vdef(_, _) -> raise (Failure ("declaring local variable is only allowed in
blocks"))
      in

    (*Only variables defined outside any block are globals*)
    let check_global_stmt = function
        Vdef(t, initds) -> List.iter
            (fun initd -> match initd with InitDectr(d, _) ->
                let tt = check_vdef StringMap.empty StringMap.empty t initd in
                let id = id_of_dectr d in
                if not (Hashtbl.mem globals id) then Hashtbl.add globals id (*fst initd*)tt
                else raise (Failure ("duplicate global " ^ id)))
            initds
      | stmt -> check_stmt StringMap.empty StringMap.empty Int stmt
    in

    List.iter check_global_stmt (List.rev statements);
    Hashtbl.iter (fun _ f -> if not f.checked then (func_checked f; check_func f) else ())
func_decls
```

8.5 codegen.ml

```
(* codegen.ml *)
(* By: Oswin, Danielle, Tyrus *)

(* easel code generation *)
module L = Llvm
module A = Ast
module StringMap = Map.Make(String)

type easel_env = {
    locals: (L.llvalue * (A.typ * A.dectr * bool)) StringMap.t;
    builder: L.llbuilder;
    the_func: L.llvalue;
    ret_typ: A.typ;
}

let translate (functions, statements) =
       let context = L.global_context() in
       let the_module = L.create_module context "easel"
       and i32_t = L.i32_type context
        and i8_t = L.i8_type   context
       and float_t = L.double_type context
       and i1_t = L.i1_type context
       and void_t = L.void_type context
       and pix_t = L.i32_type context
       and arr_t t n = L.array_type t n in
        let n_ptr_t t inner_len = function
          1 -> L.pointer_type t
```

```
        | 2 -> L.pointer_type (arr_t t inner_len)
        | _ -> raise (Failure "invalid n for n_ptr_t")
    in
    let ptr_t t = n_ptr_t t 0 1 in

    let zero = L.const_int i32_t 0 in
    let rec zero_list = function
          1 -> [zero]
        | n when n > 1 -> zero :: zero_list (n - 1)
        | _ -> raise (Failure "invalid n for zero_list")
    in
    let zero_arr n = Array.of_list (zero_list n) in

       let rec lltype_of_typ = function
           A.Int -> i32_t
         | A.Float -> float_t
         | A.Bool -> i1_t
         | A.Void -> void_t
         | A.Pix -> pix_t
          | A.Func(rt, fts) ->
              let formal_t = Array.of_list (List.map (fun ft -> lltype_of_typ ft) fts) in
              ptr_t (L.function_type (lltype_of_typ rt) formal_t)
         | A.ArrRef(A.ArrRef(t, l), _) ->
              let t' = lltype_of_typ t in
              ptr_t (arr_t t' l)
         | A.ArrRef(t, _) ->
              let t' = lltype_of_typ t in
              ptr_t t'
        in

    let rec lltype_of_dectr t = function
        A.DecArr(d, l) -> arr_t (lltype_of_dectr t d) l
      | A.DecId(_) -> lltype_of_typ t
    in

    let rec llval_of_dectr t = function
        A.DecArr(d, l) -> L.const_array (lltype_of_dectr t d) (Array.make l (llval_of_dectr t
d))
      | A.DecId(_) -> (match t with
                          A.Int -> L.const_int (lltype_of_typ t) 0
                        | A.Pix -> L.const_int (lltype_of_typ t) 0
                        | A.Bool -> L.const_int (lltype_of_typ t) 0
                        | A.Float -> L.const_float (lltype_of_typ t) 0.0
                        | _ -> raise (Failure "not a valid type for declaration")
                       )

    in

    let rec id_of_dectr = function
        A.DecId(id) -> id
      | A.DecArr(d, _) -> id_of_dectr d
    in

    let sub_dectr = function
        A.DecId(_) as d -> d
      | A.DecArr(d, _) -> d
    in
```

```
    let decarr_len = function
        A.DecArr(_, l) -> l
      | A.DecId(id) -> raise (Failure (id ^ " is not an array"))
    in

    let rec get_arr_id = function
        A.Id(id) -> id
      | A.EleAt(id,_) -> get_arr_id id
      | A.PropAcc(id,_) -> get_arr_id id
      | _ -> raise (Failure "does not have id")
    in

    let globals = Hashtbl.create 8 in

    let function_decls = Hashtbl.create 8 in
    let function_decl tbl fdecl =
        let name = fdecl.A.fname
        and formal_t = Array.of_list (List.map (fun (t,_) -> lltype_of_typ t) fdecl.A.formals)
in
        let ftype = L.function_type (lltype_of_typ fdecl.A.typ) formal_t in
        Hashtbl.add tbl name (L.define_function name ftype the_module, fdecl); tbl in
        let _ = List.fold_left function_decl function_decls functions
    in

    let anonfunc_decls = Hashtbl.create 8 in
    let anonfunc_decl fdecl =
        let name = "____ReSeRvEd_AnOnYmOuS_fUnC__" ^
                   (string_of_int ((Hashtbl.length anonfunc_decls) + 1))
        and formal_t = Array.of_list (List.map (fun (t,_) -> lltype_of_typ t) fdecl.A.formals)
in
        let ftype = L.function_type (lltype_of_typ fdecl.A.typ) formal_t in
        let fp = L.define_function name ftype the_module in
          Hashtbl.add anonfunc_decls name (fp, fdecl); fp
    in

    (* built-in functions *)
    let extfunc_draw_def_t = L.var_arg_function_type i32_t [||] in
    let extfunc_draw_def = L.declare_function "draw_default" extfunc_draw_def_t the_module in
    let extfunc_do_draw_t = L.var_arg_function_type i32_t [|ptr_t i32_t; i32_t; i32_t; i32_t;
i32_t|] in
    let extfunc_do_draw = L.declare_function "do_draw" extfunc_do_draw_t the_module in
    let extfunc_printf_t = L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
    let extfunc_printf = L.declare_function "printf" extfunc_printf_t the_module in

    let extfunc_pow_t = L.function_type float_t [| float_t; float_t |] in
    let extfunc_pow = L.declare_function "pow" extfunc_pow_t the_module in
    let pow_call b e n bdr = L.build_call extfunc_pow [|b; e|] n bdr in

    let extfunc_sin_t = L.var_arg_function_type float_t [|float_t|] in
    let extfunc_sin = L.declare_function "sin" extfunc_sin_t the_module in
    let extfunc_cos_t = L.var_arg_function_type float_t [|float_t|] in
    let extfunc_cos = L.declare_function "cos" extfunc_cos_t the_module in
    let extfunc_tan_t = L.var_arg_function_type float_t [|float_t|] in
    let extfunc_tan = L.declare_function "tan" extfunc_tan_t the_module in

    (* rand and log *)
    let extfunc_log_t = L.var_arg_function_type float_t [|float_t|] in
```

```
    let extfunc_log_t = L.var_arg_function_type float_t [|float_t|] in
    let extfunc_log = L.declare_function "log" extfunc_log_t the_module in

    let extfunc_rand_t = L.function_type float_t [||] in
    let extfunc_rand = L.declare_function "rando" extfunc_rand_t the_module in

    let extfunc_rand_t = L.function_type float_t [|i32_t|] in
    let extfunc_rands = L.declare_function "randos" extfunc_rand_t the_module in

    let rec __dectr_arr_dim n = function
        A.DecArr(d, _) -> __dectr_arr_dim (n + 1) d
      | A.DecId(_) -> n
    in

    let dectr_arr_dim d = __dectr_arr_dim 0 d in

    let lookup env n = try StringMap.find n env.locals
        with Not_found -> Hashtbl.find globals n in

    let load_var env id =
        let (var, (ty, dectr, _)) = lookup env id in
        let dim = dectr_arr_dim dectr in
        if dim = 0 then L.build_load var id env.builder
        else
        let arr_pos = L.build_in_bounds_gep var (zero_arr dim) id env.builder in
        let inner_len = match dectr with A.DecId(_) | A.DecArr(A.DecId(_), _) -> 0 |
A.DecArr(d, _) -> decarr_len d in
        let arr_ptr_t = n_ptr_t (lltype_of_typ ty) inner_len dim in
        L.build_bitcast arr_pos arr_ptr_t id env.builder
     in

       (* Constructing code for expressions *)
       let rec expr env = function
           A.IntLit i -> L.const_int i32_t i
         | A.FloatLit f -> L.const_float float_t f
         | A.BoolLit b -> L.const_int i1_t (if b then 1 else 0)
         | A.PixLit (r_e, g_e, b_e, a_e) -> let r_v = expr env r_e
                                    and g_v = expr env g_e
                                    and b_v = expr env b_e
                                    and a_v = expr env a_e in
                                    let shift_r = L.const_int i32_t 16777216 (* left shift for
24 bits*)
                                    and shift_g = L.const_int i32_t 65536 (* left shift for 16
bits*)
                                    and shift_b = L.const_int i32_t 256 in (* left shift for 8
bits*)
                                    let r_v' = L.build_mul r_v shift_r "tmp" env.builder
                                    and g_v' = L.build_mul g_v shift_g "tmp" env.builder
                                    and b_v' = L.build_mul b_v shift_b "tmp" env.builder in
                                    let p_v' = L.build_add r_v' g_v' "tmp" env.builder in
                                    let p_v'' = L.build_add p_v' b_v' "tmp" env.builder in
                                      L.build_add p_v'' a_v "tmp" env.builder
          | A.Id id -> (try load_var env id
                    with Not_found -> fst (Hashtbl.find function_decls id))
         | A.Noexpr -> L.const_int i32_t 0
         | A.Binop (e1, op, e2) ->
           let exp1 = expr env e1
```

```
            and exp2 = expr env e2 in
            let typ1 = L.string_of_lltype (L.type_of exp1)
            and typ2 = L.string_of_lltype (L.type_of exp2) in
            let build_op_by_type opf opi = (match (typ1, typ2) with
                  ("double", "double") -> opf
                | ("i32", "i32") -> opi
                | ("double", "i32") ->
                    (fun e1 e2 n bdr -> let e2' = L.build_sitofp e2 float_t n bdr in
                                        opf e1 e2' "tmp" bdr)
                | ("i32", "double") ->
                    (fun e1 e2 n bdr -> let e1' = L.build_sitofp e1 float_t n bdr in
                                        opf e1' e2 "tmp" bdr)
                | _ -> raise (Failure "not a valid type")
                ) in
            (match op with
              A.Add -> build_op_by_type L.build_fadd L.build_add
            | A.Sub -> build_op_by_type  L.build_fsub L.build_sub
            | A.Mult -> build_op_by_type L.build_fmul L.build_mul
            | A.Div -> build_op_by_type  L.build_fdiv L.build_sdiv
            | A.Mod -> build_op_by_type  L.build_frem L.build_srem
            | A.Pow -> (match (typ1, typ2) with
                          ("double", "double") -> pow_call
                        | ("double", "i32") -> (fun e1 e2 n bdr -> let e2' = L.build_sitofp e2
float_t "tmp" bdr in
                                                   pow_call e1 e2' n bdr)
                        | ("i32", "double") -> (fun e1 e2 n bdr -> let e1' = L.build_sitofp e1
float_t "tmp" bdr in
                                                   pow_call e1' e2 n bdr)
                        | ("i32", "i32") ->    (fun e1 e2 n bdr -> let e1' = L.build_sitofp e1
float_t "tmp" bdr in
                                                                      let e2' = L.build_sitofp e2
float_t "tmp" bdr in
                                                   pow_call e1' e2' n bdr)
                        | _ -> raise (Failure "not valid type for power operator")
                        )
            | A.Equal -> build_op_by_type (L.build_fcmp L.Fcmp.Oeq) (L.build_icmp L.Icmp.Eq)
            | A.Neq -> build_op_by_type (L.build_fcmp L.Fcmp.One) (L.build_icmp L.Icmp.Ne)
            | A.Less -> build_op_by_type (L.build_fcmp L.Fcmp.Olt) (L.build_icmp L.Icmp.Slt)
            | A.Leq -> build_op_by_type (L.build_fcmp L.Fcmp.Ole) (L.build_icmp L.Icmp.Sle)
            | A.Greater -> build_op_by_type (L.build_fcmp L.Fcmp.Ogt) (L.build_icmp
L.Icmp.Sgt)
            | A.Geq -> build_op_by_type (L.build_fcmp L.Fcmp.Oge) (L.build_icmp L.Icmp.Sge)
            | A.And -> L.build_and
            | A.Or -> L.build_or
            ) exp1 exp2 "tmp" env.builder
        | A.Unop(op, e) ->
              let exp = expr env e in
               let typ = L.string_of_lltype (L.type_of exp) in
              (match op with
               A.Neg -> (match typ with
                           "double" -> L.build_fneg exp "tmp" env.builder
                         | _ -> L.build_neg exp "tmp" env.builder)
              | A.Not -> L.build_not exp "tmp" env.builder
               | A.Inc -> (match typ with
                   "double" -> ignore(expr env (A.Assign(e, A.Binop(e, A.Add,
A.FloatLit(1.0))))); exp
                 | _ -> ignore(expr env (A.Assign(e, A.Binop(e, A.Add, A.IntLit(1))))); exp
```

```
                  )
                | A.Dec -> (match typ with
                    "double" -> ignore(expr env (A.Assign(e, A.Binop(e, A.Sub,
A.FloatLit(1.0))))); exp
                  | _ -> ignore(expr env (A.Assign(e, A.Binop(e, A.Sub, A.IntLit(1))))); exp
                 )
               )
          | A.Assign(e1, e2) -> let e1_id = get_arr_id e1 in
                      let (var, (_, _, fml)) = lookup env e1_id in
                      let e1' = (match e1 with
                                  A.Id _ -> var
                                  | A.EleAt(arr, ind) -> (match fml with
                                  (* if this array is not declared in formal, simply access it
in one gep *)
                                  false ->
                                      (match arr with
                                      A.Id _ -> L.build_in_bounds_gep var [|zero; expr env
ind|] e1_id env.builder
                                    | A.EleAt(_, l) -> L.build_in_bounds_gep var [|zero; expr
env ind; expr env l|] e1_id env.builder
                                    | _ -> raise (Failure "not a valid expression for array
assignment")
                                    )
                                  (* if this array is declared in formal, follow the way that
clang does *)
                                | true ->
                                      (match arr with
                                   A.Id _ -> let tmpp = L.build_load var e1_id env.builder in
                                                 L.build_in_bounds_gep tmpp [|expr env ind|]
e1_id env.builder
                                  | A.EleAt(_, l) -> let tmpp = L.build_load var e1_id
env.builder in
                                                       let tmpp = L.build_in_bounds_gep tmpp
[|expr env ind|] e1_id env.builder in
                                                         L.build_in_bounds_gep tmpp [|zero;
expr env l|] e1_id env.builder
                                    | _ -> raise (Failure "not a valid expression for array
assignment")
                                    )
                                  )
                                  | A.PropAcc(_, _) -> var (* dummy value, real work below *)
                                  | _ -> raise (Failure "not valid variable for assignment")
                                 )
                        and e2' = expr env e2 in
                          (match e1 with
                              A.PropAcc(e,s) -> let e_v = expr env e in
                                                let e_v'' = (match s with
                                 "red" -> let clr = L.const_int i32_t 16777215
                                          and shift = L.const_int i32_t 24 in
                                          let e_v' = L.build_and e_v clr "tmp" env.builder
                                          and e2_v = L.build_shl e2' shift "tmp"
env.builder in
                                            L.build_or e_v' e2_v "tmp" env.builder
                                | "green" -> let clr = L.const_int i32_t 4278255615
                                            and shift = L.const_int i32_t 16 in
                                            let e_v' = L.build_and e_v clr "tmp"
env.builder
```

```
                                                and e2_v = L.build_shl e2' shift "tmp"
env.builder in
                                                  L.build_or e_v' e2_v "tmp" env.builder
                                    | "blue" -> let clr = L.const_int i32_t 4294902015
                                                and shift = L.const_int i32_t 8 in
                                                let e_v' = L.build_and e_v clr "tmp" env.builder
                                                and e2_v = L.build_shl e2' shift "tmp"
env.builder in
                                                  L.build_or e_v' e2_v "tmp" env.builder
                                    | "alpha" -> let clr = L.const_int i32_t 4294967040 in
                                                 let e_v' = L.build_and e_v clr "tmp"
env.builder in
                                                  L.build_or e_v' e2' "tmp" env.builder
                                    | _ -> raise (Failure "not valid property access for
assignment")
                                    ) in ignore(L.build_store e_v'' var env.builder); e_v''
                                | _ -> ignore(L.build_store e2' e1' env.builder); e2'
                               )
          | A.EleAt(arr, ind) -> let id = get_arr_id arr in
              let (var, (_, _, fml)) = lookup env id in
              (match fml with
                false -> (match arr with
                  A.Id _ -> L.build_load (L.build_in_bounds_gep var [|zero; expr env ind|] id
env.builder) id env.builder
                | A.EleAt(_, l) -> L.build_load (L.build_in_bounds_gep var [|zero; expr env
ind; expr env l|] id env.builder) id env.builder
                | _ -> raise (Failure "not a valid array access")
                )
              | true -> (match arr with
                 A.Id _ -> let tmpp = L.build_load var id env.builder in
                   let tmpp = L.build_in_bounds_gep tmpp [|expr env ind|] id env.builder in
                     L.build_load tmpp id env.builder
               | A.EleAt(_, l) ->
                 let tmpp = L.build_load var id env.builder in
                 let tmpp = L.build_in_bounds_gep tmpp [|expr env ind|] id env.builder in
                 let tmpp = L.build_in_bounds_gep tmpp [|zero; expr env l|] id env.builder in
                   L.build_load tmpp id env.builder
               | _ -> raise (Failure "not a valid array access")
               )
             )
          | A.PropAcc(e,s)-> (match s with
              "red" -> let v = expr env e
                       and shift = L.const_int i32_t 24 in
                         L.build_lshr v shift "tmp" env.builder
            | "green" -> let v = expr env e
                         and shift_r = L.const_int i32_t 24
                         and shift_g = L.const_int i32_t 16 in
                         let r_v = L.build_lshr v shift_r "tmp" env.builder
                         and g_v = L.build_lshr v shift_g "tmp" env.builder
                         and shift = L.const_int i32_t 256 in
                         let r_v' = L.build_mul r_v shift "tmp" env.builder in
                           L.build_sub g_v r_v' "tmp" env.builder
            | "blue" -> let v = expr env e
                        and shift_g = L.const_int i32_t 16
                        and shift_b = L.const_int i32_t 8 in
                        let g_v = L.build_ashr v shift_g "tmp" env.builder
                        and b_v = L.build_ashr v shift_b "tmp" env.builder
```

```
                      and shift_g' = L.const_int i32_t 256 in
                      let g_v' = L.build_mul g_v shift_g' "tmp" env.builder in
                        L.build_sub b_v g_v' "tmp" env.builder
          | "alpha" -> let v = expr env e
                       and shift_b = L.const_int i32_t 8 in
                       let b_v = L.build_ashr v shift_b "tmp" env.builder
                       and shift_b' = L.const_int i32_t 256 in
                       let b_v' = L.build_mul b_v shift_b' "tmp" env.builder in
                         L.build_sub v b_v' "tmp" env.builder
          | "size" -> let id = get_arr_id e in
                      let (_, (_, dectr, _ )) = lookup env id in
                        expr env (A.IntLit (decarr_len dectr))
          | _ -> raise (Failure "not a valid property access")
          )
        | A.AnonFunc(fdecl) -> anonfunc_decl fdecl
        (* Call external functions *)
        | A.Call (A.Id("draw"), []) -> L.build_call extfunc_draw_def [||] "draw_def"
env.builder
        | A.Call (A.Id("draw"), [A.Id(cid); e2; e3]) ->
            let (c_llval, (_, c_col, _)) = lookup env cid in
            let c_row = sub_dectr c_col in
            let w = decarr_len c_row in
            let h = decarr_len c_col in
            let c_ptr = L.build_in_bounds_gep c_llval (zero_arr 3) "cnvstmp" env.builder in
              L.build_call extfunc_do_draw [| c_ptr; L.const_int i32_t w; L.const_int i32_t
h;
                                              expr env e2; expr env e3 |] "do_draw" env.builder
        | A.Call (A.Id("draw_size"), [e_c; e_w; e_h; e4; e5]) ->
            let c_llval = expr env e_c in
            let c_ptr = L.build_in_bounds_gep c_llval (zero_arr 2) "cnvstmp" env.builder in
              L.build_call extfunc_do_draw [| c_ptr; expr env e_w; expr env e_h;
                                              expr env e4; expr env e5 |] "do_draw" env.builder
        | A.Call (A.Id("print"), [e]) ->
            let int_format_str = L.build_global_stringptr "%d\n" "fmt" env.builder in
              L.build_call extfunc_printf [| int_format_str ; (expr env e) |] "printf"
env.builder
        | A.Call (A.Id("printp"), [e]) ->
            let int_format_str = L.build_global_stringptr "#%x\n" "fmt" env.builder in
              L.build_call extfunc_printf [| int_format_str ; (expr env e) |] "printf"
env.builder
        | A.Call (A.Id("printb"), [e]) ->
            let int_format_str = L.build_global_stringptr "%d\n" "fmt" env.builder in
              L.build_call extfunc_printf [| int_format_str ; (expr env e) |] "printf"
env.builder
        | A.Call (A.Id("printfl"), [e]) ->
            let float_format_str = L.build_global_stringptr "%f\n" "fffmt" env.builder in
              L.build_call extfunc_printf [| float_format_str ; (expr env e) |] "printf"
env.builder
        | A.Call (A.Id("sin"), [e]) ->
           L.build_call extfunc_sin [|expr env e|] "sin" env.builder
        | A.Call (A.Id("cos"), [e]) ->
            L.build_call extfunc_cos [|expr env e|] "cos" env.builder
        | A.Call (A.Id("tan"), [e]) ->
            L.build_call extfunc_tan [|expr env e|] "tan" env.builder
        | A.Call (A.Id("log"), [b; e]) ->
            let log_it x = L.build_call extfunc_log [|x|] "tmp_log" env.builder in
            let promote x = (let ex = expr env x in
```

```
                let typ = L.string_of_lltype (L.type_of ex) in
                (match typ with
                    "double" -> log_it ex
                  | "i32" -> (let pex = L.build_sitofp ex float_t "tmp" env.builder in
                              log_it pex)
                  | _ -> raise (Failure "not a valid type"))) in
                let base = promote b
                and sol = promote e in
                  L.build_fdiv (sol) (base) "log" env.builder
          | A.Call (A.Id("rand"), []) -> L.build_call extfunc_rand [||] "rand_call"
env.builder
          | A.Call (A.Id("rand"), [e]) -> L.build_call extfunc_rands [|(expr env e)|]
"rand_call" env.builder
          | A.Call (A.Id(func), act) ->
            let fdef = expr env (A.Id(func)) in
            let ret_t = L.string_of_lltype (L.return_type (L.return_type (L.type_of fdef))) in
            let ret_n = match ret_t with
                "void" -> ""
              | _ -> "tmp" in
            let actuals = List.rev (List.map (expr env) (List.rev act)) in
              L.build_call fdef (Array.of_list actuals) ret_n env.builder
          | A.Call(_,_) -> raise (Failure "not a valid function call")
    in

    let init_var t dectr env = function
        A.IntLit i -> expr env (A.IntLit i)
      | A.FloatLit f -> expr env (A.FloatLit f)
      | A.BoolLit b -> expr env (A.BoolLit b)
      | A.PixLit (r, g, b, a) -> expr env (A.PixLit (r,g,b,a))
      | A.Binop (e1, op, e2) -> expr env (A.Binop(e1,op,e2))
      | _ -> llval_of_dectr t dectr
    in

    let global_var t env = function A.InitDectr(dectr, init) ->
      let inst = llval_of_dectr t dectr in
      let n = id_of_dectr dectr in
        Hashtbl.add globals n (L.define_global n inst the_module, (t, dectr, false));
      if init != A.Noexpr then
        match dectr with
        (* Only store initial values for scalar variables *)
          A.DecId(id) -> ignore (expr env (A.Assign(A.Id(id), init)))
        | A.DecArr(_, _) -> ()
      else ()
    in

    let local_var t env = function A.InitDectr(dectr, init) ->
      let n = id_of_dectr dectr in
      let loc = L.build_alloca (lltype_of_dectr t dectr) n env.builder in
      let _ = match dectr with
                (* Only store initial values for scalar variables *)
                A.DecId(_) -> L.build_store (init_var t dectr env init) loc env.builder
                (* loc is returned as some meaningless dummy value *)
              | A.DecArr(_, _) -> loc in
      let locals = StringMap.add n (loc, (t, dectr, false)) env.locals in
      { env with locals = locals }
    in
```

```
    (* Invoke "f builder" if the current block doesn't already
       have a terminal (e.g., a branch). *)
    let add_terminal builder f =
      match L.block_terminator (L.insertion_block builder) with
       Some _ -> ()
      | None -> ignore (f builder) in

    let rec stmt env = function
        (* Discard the locals built in the inner block *)
        A.Block sl -> List.fold_left stmt env sl
      | A.Expr e -> ignore (expr env e); env
      | A.Vdef (t, initds) ->
          List.fold_left (local_var t) env (List.rev initds)
      | A.If (pred, then_stmt, else_stmt) ->
          let bool_val = expr env pred in
         let merge_bb = L.append_block context "merge" env.the_func in

         let then_bb = L.append_block context "then" env.the_func in
         let then_env = { env with builder = (L.builder_at_end context then_bb) } in
          add_terminal (stmt then_env then_stmt).builder
          (L.build_br merge_bb);

          let else_bb = L.append_block context "else" env.the_func in
         let else_env = { env with builder = (L.builder_at_end context else_bb) } in
         add_terminal (stmt else_env else_stmt).builder
         (L.build_br merge_bb);

         ignore (L.build_cond_br bool_val then_bb else_bb env.builder);
         { env with builder = (L.builder_at_end context merge_bb) }
      | A.While (pred, body) ->
        let pred_bb = L.append_block context "while" env.the_func in
        ignore (L.build_br pred_bb env.builder);

        let body_bb = L.append_block context "while_body" env.the_func in
        let body_env = { env with builder = (L.builder_at_end context body_bb) } in
        add_terminal (stmt body_env body).builder
        (L.build_br pred_bb);

        let pred_env = { env with builder = (L.builder_at_end context pred_bb) } in
        let pred_v = expr pred_env pred in
        let pred_t = L.type_of pred_v in

        let merge_bb = L.append_block context "merge" env.the_func in
        let cmp_v =
            if pred_t = i32_t then L.build_icmp L.Icmp.Ne pred_v (L.const_int i32_t 0) "cmp"
pred_env.builder
            else pred_v in
        ignore (L.build_cond_br cmp_v body_bb merge_bb pred_env.builder);
        { env with builder = (L.builder_at_end context merge_bb) }

      | A.For (e1, e2, e3, body) -> stmt env
        ( A.Block [A.Expr e1 ; A.While (e2, A.Block [body ; A.Expr e3]) ] )

      | A.Return e -> let e' = expr env e in
        let e_t = L.type_of e' in
                    ignore (match (env.ret_typ, e_t) with
                      (A.Void, _) -> L.build_ret_void env.builder
```

```
              | (A.Int, _) -> let e'' = L.build_fptosi e' i32_t "tmp" env.builder in
                                L.build_ret e'' env.builder
              | (A.Float, _) -> let e'' = L.build_sitofp e' float_t "tmp" env.builder in
                                L.build_ret e'' env.builder
                    | _ -> L.build_ret e' env.builder
           ); env
    in

    let global_stmt env = function
          A.Vdef(t, initds) -> List.iter (global_var t env) (List.rev initds); env
        | st -> stmt env st
    in

    let build_main_function sl =
        (* Define the main function for executing global statements *)
        let ftype_main = L.function_type i32_t [||] in
        let main_func = L.define_function "main" ftype_main the_module in
        let builder = L.builder_at_end context (L.entry_block main_func) in
        let env = { locals = StringMap.empty; builder = builder; the_func = main_func; ret_typ
= A.Int } in

        let end_env = List.fold_left global_stmt env sl in

        (* Add a return if the last block falls off the end *)
        add_terminal end_env.builder (L.build_ret (L.const_int i32_t 0))
    in

    let build_function_body _ (the_function, fdecl) =
        let builder = L.builder_at_end context (L.entry_block the_function) in
        let add_formal m (ty, dectr) p =
            let n = id_of_dectr(dectr) in
              L.set_value_name n p;
            let local = L.build_alloca (lltype_of_typ ty) n builder in
              ignore (L.build_store p local builder);
            (* the third field indicates whether it is a formal *)
            StringMap.add n (local, (ty, dectr, true)) m
        in
        let formals = List.fold_left2 add_formal StringMap.empty fdecl.A.formals
            (Array.to_list (L.params the_function))
        in
        let env = { locals = formals; builder = builder; the_func = the_function; ret_typ =
fdecl.A.typ } in

        (* Build the code for each statement in the function *)
        let env = stmt env (A.Block fdecl.A.body) in

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

    build_main_function (List.rev statements);
    Hashtbl.iter build_function_body function_decls;
    Hashtbl.iter build_function_body anonfunc_decls;
    the_module
```

8.6 easel.ml

```
(* easel.ml *)
(* By: Oswin, Danielle, Sophie, Tyrus *)

(* Top-level of the easel compiler: scan & parse the input,
   check the resulting AST, generate LLVM IR, and dump the module *)

type action = Ast | LLVM_IR | Compile

let _ =
  let action = if Array.length Sys.argv > 1 then
      List.assoc Sys.argv.(1) [ ("-a", Ast);         (* Print the AST only *)
                                ("-l", LLVM_IR);  (* Generate LLVM, don't check *)
                                ("-c", Compile) ] (* Generate, check LLVM IR *)
    else Compile in
  let lexbuf = Lexing.from_channel stdin in
  let ast = Parser.program Scanner.token lexbuf in
  Semant.check ast;
  match action with
    Ast -> print_string (Ast.string_of_program ast)
  | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate ast))
  | Compile -> let m = Codegen.translate ast in
    Llvm_analysis.assert_valid_module m;
    print_string (Llvm.string_of_llmodule m)
```

8.7 autotest.sh

```
# autotest.sh
# By: Oswin, Sophie
#!/bin/sh

# Regression testing script for easel
# Step through a list of files
#  Compile, run, and check the output of each expected-to-work test
#  Compile and check the error of each expected-to-fail test

# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the easel compiler.  Usually "./easel.native"
# Try "_build/easel.native" if ocamlbuild was unable to create a symbolic link.
EASEL="./easel.native"
#EASEL="_build/easel.native"

# Set time limit for all operations
ulimit -t 30

globallog=autotest.log
rm -f $globallog
error=0
globalerror=0

keep=0
ast=0
```

```
Usage() {
    echo "Usage: autotest.sh [options] [.es files]"
    echo "-k    Keep intermediate files"
    echo "-a    Test AST output"
    echo "-h    Print this help"
    exit 1
}

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

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

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

# RunFail <args>
# Report the command, run it, and expect an error
RunFail() {
    echo $* 1>&2
    eval $* && {
       SignalError "failed: $* did not report an error"
       return 1
    }
    return 0
}

Check() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
                              s/.es$//'`
    reffile=`echo $1 | sed 's/.es$//'`

    echo -n "$basename..."
    echo 1>&2
    echo "###### Testing $basename" 1>&2
```

```
    generatedfiles=""

    if [ $ast -eq 1 ]; then
        generatedfiles="$generatedfiles ${basename}.ast" &&
        Run "$EASEL -a" "<" $1 ">" "${basename}.ast" &&
        Compare ${basename}.ast ${reffile}.ast ${basename}.diff
    else
        generatedfiles="$generatedfiles ${basename}.ll ${basename}.out" &&
        Run "$EASEL" "<" $1 ">" "${basename}.ll" &&
        Run "$LLI" "${basename}.ll" ">" "${basename}.out" &&
        Compare ${basename}.out ${reffile}.out ${basename}.diff
    fi

    # Report the status and clean up the generated files

    if [ $error -eq 0 ] ; then
       if [ $keep -eq 0 ] ; then
           rm -f $generatedfiles
       fi
       echo "OK"
       echo "###### SUCCESS" 1>&2
    else
       echo "###### FAILED" 1>&2
       globalerror=$error
    fi
}

CheckFail() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
                            s/.es$//'`
    reffile=`echo $1 | sed 's/.es$//'`

    echo -n "$basename..."

    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    if [ $ast -eq 1 ]; then
        generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&
        RunFail "$EASEL -a" "<" $1 "2>" "${basename}.err" ">>" $globallog &&
        Compare ${basename}.err ${reffile}.err ${basename}.diff
    else
        generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&
        RunFail "$EASEL" "<" $1 "2>" "${basename}.err" ">>" $globallog &&
        Compare ${basename}.err ${reffile}.err ${basename}.diff
    fi

    # Report the status and clean up the generated files

    if [ $error -eq 0 ] ; then
       if [ $keep -eq 0 ] ; then
           rm -f $generatedfiles
       fi
       echo "OK"
```

```
        echo "###### SUCCESS" 1>&2
    else
        echo "###### FAILED" 1>&2
        globalerror=$error
    fi
}

while getopts "hka" c; do
    case $c in
        k) # Keep intermediate files
            keep=1
            ;;
        a) # Test the pretty-printing of AST
            ast=1
            ;;
        h) # Help
            Usage
            ;;
    esac
done

shift `expr $OPTIND - 1`
# Parameters appearing after double dash are positional parameters
[ "$1" = "--" ] && shift

if [ $# -ge 1 ]
then
    files=$@
else
    files="tests/test-*.es tests/fail-*.es"
fi

cd glwrap
make
cd ..
make

for file in $files
do
    case $file in
        *test-*)
            Check $file 2>> $globallog
            ;;
        *fail-*)
         CheckFail $file 2>> $globallog
            ;;
        *)
            echo "unknown file type $file"
            globalerror=1
            ;;
    esac
done

#./ecc.sh -l hello.es
#./hello

exit $globalerror
```

8.8 Makefile

```
# Make sure ocamlbuild can find opam-managed packages: first run
#
# eval `opam config env`

# Easiest way to build: using ocamlbuild, which in turn uses ocamlfind

.PHONY : easel.native

easel.native :
	ocamlbuild -use-ocamlfind -pkgs llvm,llvm.analysis -cflags -w,+a-4 \
		easel.native

# "make clean" removes all generated files

.PHONY : clean
clean :
	ocamlbuild -clean
	rm -rf autotest.log *.diff easel scanner.ml parser.ml parser.mli
	rm -rf *.cmx *.cmi *.cmo *.cmx *.o *.output *.ast *.err *.diff *.ll *.out

# More detailed: build using ocamlc/ocamlopt + ocamlfind to locate LLVM

OBJS = ast.cmx codegen.cmx parser.cmx scanner.cmx semant.cmx easel.cmx

easel : $(OBJS)
	ocamlfind ocamlopt -linkpkg -package llvm -package llvm.analysis $(OBJS) -o easel

scanner.ml : scanner.mll
	ocamllex scanner.mll

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

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

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

%.cmx : %.ml
	ocamlfind ocamlopt -c -package llvm $<

### Generated by "ocamldep *.ml *.mli" after building scanner.ml and parser.ml
ast.cmo :
ast.cmx :
codegen.cmo : ast.cmo
codegen.cmx : ast.cmx
easel.cmo : semant.cmo scanner.cmo parser.cmi codegen.cmo ast.cmo
easel.cmx : semant.cmo scanner.cmx parser.cmx codegen.cmx ast.cmx
parser.cmo : ast.cmo parser.cmi
parser.cmx : ast.cmx parser.cmi
scanner.cmo : parser.cmi
scanner.cmx : parser.cmx
semant.cmo : ast.cmo
semant.cmx : ast.cmx
parser.cmi : ast.cmo
```

```
# Building the tarball

GLWRAP = $(filter-out glwrap/_build, $(wildcard glwrap/*))

TESTFILES = $(wildcard tests/test-*.es) $(wildcard tests/test-*.ast) \
                     $(wildcard tests/test-*.out) $(wildcard tests/fail-*.es) \
                     $(wildcard tests/fail-*.err)

TARFILES = ast.ml codegen.ml Makefile easel.ml parser.mly README.md scanner.mll \
                 semant.ml autotest.sh $(GLWRAP) demo $(TESTFILES)

easel.tar.gz: $(TARFILES)
       cd .. && tar czf easel/easel.tar.gz $(TARFILES:%=easel/%) && cd -
```

8.9 glwrap/glwrap.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>
/*#include <GL/glew.h>*/
#ifdef __APPLE__
#  include <GLUT/glut.h>
#else
#  include <GL/glut.h>
#endif

int do_draw(int *canvas, int w, int h, int x, int y);
void render(void);
void myglinit(void);
double rando();
double randos();


int *easel;
int W, H;

int draw_default() {
#define DW 960
#define DH 960
    int c[DW][DH];
    int x, y;
    for (x = 0; x < DW; x++) {
        for (y = 0; y < DH; y++) {
            if (x > DW/2 && y > DH/2)
                c[x][y] = 0xffffff00; // RGBA
            else
                c[x][y] = 0x00000000;
        }
    }
    do_draw((int *) c, DW, DH, 0, 0);
}
```

```
int do_draw(int *canvas, int w, int h, int x, int y) {
    char *fake_argv[1];
    int fake_argc = 1;
    fake_argv[0] = strdup("easel");

    easel = canvas;
    W = w;
    H = h;

    // initialize the glut system and create a window
    glutInitWindowSize(W, H);
    glutInitWindowPosition(x, y);
    glutInit(&fake_argc, fake_argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);
    glutCreateWindow(fake_argv[0]);

    // initialize some OpenGL state, some might be redundant
    myglinit();

    // set callback functions.
    glutDisplayFunc(&render);

    // start the main glut loop, no code runs after this
    glutMainLoop();

    return 0;
}

void render(void) {
#ifdef _DEBUG
    printf("in render\n");
#endif // _DEBUG

#ifdef _DEBUG_VERB
    int x, y;
    for (y = 0; y < H; y++) {
        for (x = 0; x < W; x++) {
            printf("%d ", easel[(y * W + x)]);
        }
        printf("\n");
    }
#endif // _DEBUG_VERB

    // drawpixels draws the rgb data stored in 'easel' to the screen
    glDrawPixels(W, H, GL_RGBA, GL_UNSIGNED_INT_8_8_8_8, easel);

    // in double buffer mode so we swap to avoid a flicker
    glutSwapBuffers();

    // instruct event system to call 'render' again
    // glutPostRedisplay();
}

// set some OpenGL state variables
void myglinit() {
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
```

```
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);
}

double randos(int seed) {
    srand(seed);
    double r = rand();
    return (double) (r/RAND_MAX);
}

double rando() {
    return randos(time(NULL));
}
```

8.10 glwrap/Makefile

```
OUT_DIR = _build
APP_DIR = apps

SRC = $(wildcard *.c)
CAPPS = $(wildcard $(APP_DIR)/*.c)
LLAPPS = $(wildcard $(APP_DIR)/*.ll)
APPS = $(CAPPS:.c=) $(LLAPPS:.ll=)
OBJ = $(SRC:.c=.o)
LIBNAME = glwrap

TARGETLIB = $(OUT_DIR)/lib$(LIBNAME).a
TARGETAPPS = $(foreach app, $(notdir $(APPS)), $(OUT_DIR)/$(app))

GLFLAGS = -L$(OUT_DIR) -l$(LIBNAME) -lGL -lGLEW -lglut -lm

.PHONY: all dir clean

TARGET = $(TARGETLIB) $(TARGETAPPS)

all: dir $(TARGET)

dir: ${OUT_DIR}

${OUT_DIR}:
	mkdir -p $@

$(TARGETLIB): $(OUT_DIR)/$(OBJ)
	$(AR) rcs $@ $^

$(OUT_DIR)/%.o: %.c
	$(CC) -c -o $@ $^

$(OUT_DIR)/%: $(APP_DIR)/$(notdir %.c) $(TARGETLIB)
	gcc -o $@ $< $(GLFLAGS)

$(OUT_DIR)/%: $(APP_DIR)/$(notdir %.ll) $(TARGETLIB)
	llc -o $@.s $<
	gcc -o $@ $@.s $(GLFLAGS)
```

```
clean:
	$(RM) -rf $(OUT_DIR)
```

8.11 Test Suite Files

This is the listing of all files in our test suite and demo/ directory.

fail-assign1.err
fail-assign1.es
fail-assign2.err
fail-assign2.es
fail-assign3.err
fail-assign3.es
fail-assign4.err
fail-assign4.es
fail-assign5.err
fail-assign5.es
fail-assign6.err
fail-assign6.es
fail-assign7.err
fail-assign7.es
fail-expr1.err
fail-expr1.es
fail-expr10.err
fail-expr10.es
fail-expr2.err
fail-expr2.es
fail-expr3.err
fail-expr3.es
fail-expr4.err
fail-expr4.es
fail-expr5.err
fail-expr5.es
fail-expr6.err
fail-expr6.es
fail-expr7.err
fail-expr7.es
fail-expr8.err
fail-expr8.es
fail-expr9.err
fail-expr9.es
fail-fun1.err
fail-fun1.es
fail-fun2.err
fail-fun2.es
fail-fun3.err
fail-fun3.es
fail-local.err
fail-local.es
fail-stat.err
fail-stat.es
fail-stmt1.err
fail-stmt1.es
fail-stmt2.err
fail-stmt2.es
fail-stmt3.err
fail-stmt3.es
fail-type1.err
fail-type1.es
fail-type2.err
fail-type2.es
fail-type3.err
fail-type3.es
fail-type4.err
fail-type4.es
fail-type5.err
Fail-type5.es
demo/jset.es
demo/hello.es
demo/mandelbrot.es
demo/mandelbrot2.es
demo/mandelbrot_anon.es
test-anonfunc1.ast
test-anonfunc1.es
test-anonfunc1.out
test-anonfunc2.ast
test-anonfunc2.es
test-anonfunc2.out
test-arith-difftype1.ast
test-arith-difftype1.es
test-arith-difftype1.out
test-arith-difftype2.ast
test-arith-difftype2.es
test-arith-difftype2.out
test-arith.ast
test-arith.es
test-arith.out

test-arith2.ast
test-arith2.es
test-arith2.out
test-arr-args1.ast
test-arr-args1.es
test-arr-args1.out
test-arr-args2.ast
test-arr-args2.es
test-arr-args2.out
test-array1.ast
test-array1.es
test-array1.out
test-array2.ast
test-array2.es
test-array2.out
test-array3.ast
test-array3.es
test-array3.out
test-cirfun.ast
test-cirfun.es
test-cirfun.out
test-const.ast
test-const.es
test-const.out
test-fib1.ast
test-fib1.es
test-fib1.out
test-floatlit1.ast
test-floatlit1.es
test-floatlit1.out
test-for1.ast
test-for1.es
test-for1.out
test-for2.ast
test-for2.es
test-for2.out
test-fun-add.ast
test-fun-add.es
test-fun-add.out
test-fun-ptr.ast
test-fun-ptr.es
test-fun-ptr.out
test-global1.ast
test-global1.es
test-global1.out
test-global2.ast
test-global2.es
test-global2.out
test-if1.ast
test-if1.es
test-if1.out
test-if2.ast
test-if2.es
test-if2.out
test-if3.ast
test-if3.es
test-if3.out
test-local-arr1.ast
test-local-arr1.es
test-local-arr1.out
test-pixlit.ast
test-pixlit.es
test-pixlit.out
test-return1.ast
test-return1.es
test-return1.out
test-while1.ast
test-while1.es
test-while1.out
test-while2.ast
test-while2.es
test-while2.out