JEo-MC Final Report

A Geometric Programming Language

Jeremy Lu - jl5921, **Emma Schwartz** - es3962, **Melody Hsu** - mh4133, **Connie Zhou** - cz2382

Fall 2021

Contents

1 Introduction **4**

2 Setup **5**

2.1 Environment setup 5

2.1.1 MacOS OpenGL installation 5

2.1.2 Ubuntu OpenGL installation 6

2.2 Building JEo-MC 6

2.3 Running and Testing 6

2.4 Language Tutorial 7

3 Language Reference Manual **7**

3.1 Programming Conventions (i.e. indentation, whitespace, comments) 8

3.1.1 Identifiers 8

3.1.2 Whitespace 8

3.1.3 Comments 8

3.2 Primitives 9

3.2.1 Integer Constants 9

3.2.2 Floating Constants 9

3.2.3 Boolean Constants 10

3.2.4 String Literals 10

3.3 Reserved Keywords 10

3.3.1 Statements and Blocks 10

3.3.2 Control Flow 10

3.3.3 Types 10

3.3.3 Built-in Functions 10

3.4 Operators 11

3.4.1 Precedence 11

3.4.2 Grammar Rules for Operators 11

3.5 Functions Available to User 12

3.5.1 General function declaration 12

3.5.2 Built-In Functions 12

3.5.2.1 drawCircle() 12

3.5.2.2 drawTriangle() 12

3.5.2.3 drawRectangle() 12

3.5.2.4 drawLine() 13

3.5.2.5 jeomcInit() 13

3.5.2.6 jeomcRunAndSave() 13

3.5.3 Hexcode support 13

3.6 Program Structure (loop syntax, if/else statements) 14

3.6.1 if and else statements 14

3.6.2 while Statement 14

3.6.3 for Statement 14
3.6.4 break, continue, and return Statements 15
3.6.5 Note on Loop Control 15
3.7 Exceptions 15
3.7.1 Keyword termination 15
3.7.2 Function arguments 15
3.7.3 Binary Operators and Assignment 15
3.7.4 Duplicate functions and variables 16
3.7.5 Unrecognized or undefined functions and variables 16
3.7.6 General parsing errors 16

4 Project Plan 16
4.1 Planning Process 16
4.2 Project Timeline 16
4.3 Development Environment 17
4.4 Programming Style Guide 17
4.5 Project Log 19

5 Architectural Design 20
5.1 Block Diagram 20
5.2 Description of Interfaces between Components 20
5.2.1 Source Code 20
5.2.2 Lexer 20
5.2.3 Parser and AST 20
5.2.4 Semantic Checker 21
5.2.5 Code Generation 21
5.2.6 LLVM 21
5.2.7 C Libraries - jeomc_draw.c 21
5.2.8 Linking 21
5.3 Implementation Authors 21
5.3.1 Jeremy 21
5.3.2 Emma 22
5.3.3 Melody 22
5.3.4 Connie 22

6 Test Plan 22
6.1 Testing Suite and Integration 22
6.2 Unit and Integration Testing 23
6.3 Test Automation 23
6.4 Test Authors 23
6.4.1 Jeremy 23
6.4.2 Emma 24
6.4.3 Melody 24

6.4.4 Connie 25
6.5 Examples of Source Language Programs 25

7 Example Test Programs 28

8 Lessons Learned 28
8.1 Jeremy 28
8.2 Emma 28
8.3 Melody 29
8.4 Connie 29

9 Appendix 30
9.1 Compiler 30
9.1.1 parser.mly 30
9.1.2 scanner.mll 33
9.1.3 ast.ml 34
9.1.4 sast.ml 37
9.1.5 semant.ml 39
9.1.6 codegen.ml 45
9.2 Linker C program 52
9.2.1 jeomc_draw.c 52
9.3 Build Scripts 61
9.3.1 testall.sh 61
9.3.2 demo.sh 67
9.3.2 Makefile 71
9.4 Tests 72
9.4.1 fail-break.j 72
9.4.2 fail-continue.j 72
9.4.3 fail-draw-func1.j 73
9.4.4 fail-draw-func2.j 73
9.4.5 fail-draw-func3.j 74
9.4.6 fail-string.j 75
9.4.7 fail-string-int-comparison.j 75
9.4.8 test-arith4.j 75
9.4.9 test-arith5.j 75
9.4.10 test-bool.j 76
9.4.11 test-bool2.j 76
9.4.12 test-bool3.j 77
9.4.13 test-break.j 77
9.4.14 test-break2.j 78
9.4.15 test-continue.j 78
9.4.16 test-continue2.j 78
9.4.17 test-draw1.j 79

9.4.18 test-draw2.j 79
9.4.19 test-empty-string.j 80
9.4.20 test-for-add.j 80
9.4.21 test-for-and.j 80
9.4.21 test-for-div.j 81
9.4.22 test-for-minus.j 81
9.4.23 test-for-mul.j 82
9.4.24 test-hello2.j 82
9.4.25 test-hex1.j 82
9.4.26 test-hex2.j 83
9.4.27 test-hex3.j 83
9.4.28 test-hex4.j 84
9.4.29 test-string.j 84
9.4.30 test-string-case.j 85
9.4.31 test-string2.j 86
9.4.32 test-string3.j 86
9.5 Git Log 87

1 Introduction

JEo-MC is a high-level, simple geometric programming function that allows an end user to generate modular graphics using basic shapes. It is named after its authors: **J**eremy, **E**mma, **M**elody, **C**onnie. The user has a variety of built-in functions that the user can stack to generate complex images. At runtime, the output is automatically rendered in a new graphical window and separately saved in a PNG format to the user's active directory. JEo-MC's syntax borrows heavily from C for basic functionalities, such as variable assignment, loop structure, function definition, and basic arithmetic.

JEo-MC is primarily used to generate images of similar flavor to ones that might be generated using a simple native paint program, like Microsoft Paint. The intended audience is casual users and beginner programmers who may want the speed of an advanced graphics library like OpenGL, but who do not need the granularity of OpenGL's functions or the same breadth of features. The priority for our language was to strike a balance between offering ample functionality to create a wide range of graphics, while remaining easy to use for a casual programmer.

Our source code for the whole project can be found at this GitHub repository.

2 Setup

2.1 Environment setup

JEo-MC is designed to be cross-compatible with most macOS versions and Ubuntu. For the compiler itself, the end user will need a working installation of OCaml 4.13.1, ocamlbuild, opam, CC/GCC (on macOS, the clang compiler, version 12.0.0, is sufficient and can be installed via Xcode), and LLVM (LLVM 8.0.0 is recommended and can be installed using homebrew). Opam, the package manager for OCaml, will also be useful, and the following packages will need to be downloaded via opam:

- llvm (the same version as was installed via homebrew, or whatever method was used to install llvm previously)
- ocaml-find

For the graphics to render, the user will also need to install OpenGL.

2.1.1 MacOS OpenGL installation

OpenGL (version 3+) is provided with Xcode. Note that the version of OpenGL that is used by JEo-MC is considered “modern”; some legacy syntax is not supported, especially for shaders used in OpenGL. Finally, to complete installation of all graphics libraries, we recommend using homebrew to install the following packages:

- glad
- glew (enables forward-compatibility for some deprecated legacy OpenGL functions)
- glfw

2.1.2 Ubuntu OpenGL installation

- glfw

  First download glfw source package or clone from their repo glfw

  To compile glfw for X11 / Wayland, install the development packages
  `sudo apt install xorg-dev`

  Generate build files
  `cmake -S path/to/glfw -B path/to/build`

  Compile the library
  `cd path/to/build`
  `make`
  `make all`

  For more details check Compiling GLFW

- Glew

  ```
  sudo apt-get update -y
  sudo apt-get install -y libglew-dev
  ```

2.2 Building JEo-MC

To create a file in JEo-MC, make a file with a .j extension. Within that .j extension file, you may write source code in JEo-MC as long as you follow the language specifications and conventions provided in the LRM. After writing the source code, save the file into ./demo and run it with the command below.

```
$ make demo
```

2.3 Running and Testing

Build compiler binary by running:
```
$ make all
```

Occasionally run this command if errors are inconsistent with the code:
```
$ make clean
```

To run a test, run this command:
```
$ make test
```

To run a demo:
```
$ make demo
```

If permission is denied when you run tests or demo, give read and executable to self:
```
$ chmod 755 testall.sh or chmod 755 demo.sh
```

To debug your program, we have specific print statements that can be included to help:
`prints()` - strings
`printf()` - floats
`print()` - literals

2.4 Language Tutorial

To create a working source code for JEoMC, you must follow our language reference manual to create valid syntax. Every file containing source code that needs to be run needs an entry point `main()` function.

In the main function, you can then use different features of our language. Please note that in order to draw anything, you must always start a function with jeomcInit() to set up the window, and jeomcRunAndSave() to keep the window open and save the file at the end. Any number of function calls, variable declarations, and other code logic may be added in between.

```
int main()
{
	jeomcInit();

	drawCircle(-0.2,-0.2,0.25,"#FFDE00");

	jeomcRunAndSave();
```

```
	return 0;
}
```

3 Language Reference Manual

3.1 Programming Conventions (i.e. indentation, whitespace, comments)

3.1.1 Identifiers

An identifier in JEo-MC is a name given to a variable, method, interface, package, or class by the user. Each identifier must be unique, cannot conflict with a preexisting built-in function, and cannot be used more than once. An identifier must begin with a letter, followed by either numbers, letters, or underscores. There is no preference in using uppercase or lowercase letters. There are specific naming conventions associated with identifiers, represented by the regular expression below.

```
['a' - 'z' 'A' - 'Z']['a' - 'z' 'A' - 'Z' '0' - '9' '_']*
```

3.1.2 Whitespace

In JEo-MC, whitespace is not syntactically important if the rest of the statement is semantically valid. Whitespace is defined as any space character, including the tab character. However, it is good practice for statements to not be written on a single line and instead be written on multiple lines, as it is easier to comprehend. It is also good practice to pad binary operators (i.e., =, +) with a space on either side. Between unary operators, note that there cannot be whitespace (this is an explicit requirement). Tabs are suggested to clearly specify in-function code, where each subsequent scope depth increases the tab amount. Tabs should be equivalent to 4 spaces. A good example of whitespace use can be found below.

```
<part_of_method>
	<rest_of_method>
	bool i;
	i = i + 1;
	return;
```

3.1.3 Comments

JEo-MC supports block and single line comments. The syntax for any type of comment is given by /* */. Comments are encouraged, including e.g. block comments at the header of a file, or inline comments to delineate different parts of a more complex function call. Inline comments should be at the same level of indentation as the code they are commenting on.

Single line comment example:

```
a = true; /* a is true */
```

Block comment example:

```
Int main(){
/*
      This program will print out hello world

*/
}
```

3.2 Primitives

Primitives are supported by JEo-MC, including integer constants, floating constants, boolean constants, and string literals.

3.2.1 Integer Constants

An int is a signed fixed-point primitive. It is 4 bytes and values range from `-2,147,483,648` to `2,147,483,647`. Note that we do not prevent a user from declaring an int outside of those bounds, but behavior will be unpredictable (the int will likely wrap around).

Examples of Integers: `0, 90, 293`

How to declare/define Integers in JEo-MC:

```
int b;
b = 10;
```

Context Free Grammar for an integer literal(found in parser.mly) is `expr: LITERAL`

3.2.2 Floating Constants

A float is a signed floating-point primitive with precision up to 15 decimal places. Our float is more similar to traditional C's double than its float, as we allocate 8 bytes storage with a value range of `2.3E-308` to `1.7E+308`. Again we do not prevent a user from declaring a float outside of these bounds, but behavior will be unpredictable.

Examples of Floats: `2.0, 3.4, 2.3e+8`

How to declare/define Floats in JEo-MC:

```
float c;
c = 4.4;
```

Context Free Grammar for a float literal is: `expr: FLIT`

3.2.3 Boolean Constants

A boolean is either `true` or `false.`

How to declare/define Booleans in JEo-MC:

```
bool d;
d = false;
```

Context Free Grammar for a boolean literal is `expr: BLIT`

3.2.4 String Literals

A string literal is an array or sequence of characters surrounded by double quotes. For ease of use, we give our string literals a primitive type of its own, `string.` In JEo-MC, strings are compared by value when you use the equality operator `==`.

Examples of Strings: `“HELLO WORLD”, “aBcDbe”, “jeomc”`

How to declare/define Strings in JEo-MC:

```
string e;
e = “hello world”;
```

Context Free Grammar for a string literal is `expr: SLIT`

3.3 Reserved Keywords

JEo-MC has a specific set of reserved keywords that may not be used as identifiers.

3.3.1 Statements and Blocks

The following keywords specify types of statements or blocks.

```
def
```

3.3.2 Control Flow

The following keywords specify types used for control flow.

```
if then else for while continue break return
```

3.3.3 Types

Primitive types are specified by specific names used for declaration.

```
int float bool string
```

3.3.3 Built-in Functions

Pre-defined function names used for drawing

```
jeomcInit, drawRectangle, drawTriangle, drawCircle, drawLine,
jeomcSaveAndRun
```

3.4 Operators

3.4.1 Precedence

Operator	Description	Associativity
- !	Unary minus Logical NOT and bitwise NOT	Right-to-left
* / %	Multiplication, division, and mod	Left-to-right
+ -	Addition and subtraction	Left-to-right
< <= > >=	For relational operators < and ≤ For relational operators > and ≥	Left-to-right
== !=	For relational operators = and ≠	Left-to-right
&&	Logical AND	Left-to-right
||	Logical OR	Left-to-right
=	Simple assignment	Right-to-left

3.4.2 Grammar Rules for Operators

The list displays the grammar rules for operator expressions below:

```
expr:
     !expr
     -expr
     expr * expr
     expr / expr
```

```
expr % expr
expr + expr
expr - expr
expr < expr
expr <= expr
expr > expr
expr >= expr
expr == expr
expr != expr
expr && expr
expr || expr
```

3.5 Functions Available to User

3.5.1 General function declaration

General function declaration is defined by the following syntax:

```
<returntype> fun name(<type1> arg1, <type2> arg2...):
...
    return 0;
```

3.5.2 Built-In Functions

JEo-MC has an extensive repertoire of built-in functions which primarily consist of drawing functions. Users are able to make use of the following functions:

3.5.2.1 drawCircle()

```
void drawCircle(float x, float y, float radius, string color);
```

- The `drawCircle` function draws a filled-in circle centered at (x,y) in the coordinate plane with radius radius.

3.5.2.2 drawTriangle()

```
void drawTriangle(float x1, float y1, float x2, float, y2, float x3,
float y3, string color);
```

- The `drawTriangle` function draws a filled-in triangle with coordinates at (x1,y1), (x2,y2), and (x3,y3). Drawing is executed in the order of the coordinates provided.

3.5.2.3 drawRectangle()

```
void drawRectangle(float x1, float y1, float x2, float y2, float x3,
float y3, float x4, float y4, string color);
```

- The `drawRectangle` function draws a filled-in rectangle with coordinates at (x1,y1), (x2,y2), (x3,y3), and (x4,y4). Drawing is executed in the order of the coordinates provided; thus, care must be taken to ensure that the user does not "cross" the coordinates, which may result in unexpected output.

3.5.2.4 drawLine()

```
void drawLine(float x1, float y1, float x2, float y2, string color);
```

- The `drawLine` function draws a single-pixel-width line between (x1,y1) and (x2,y2). We recommend calling drawLine last in any JEo-MC program, because of the way that OpenGL draws lines - it puts the GLFW engine in "polygon mode," which switches to drawing outlines rather than filled shapes. Placing drawLine calls elsewhere in the program may lead to unexpected output.

3.5.2.5 jeomcInit()

```
void jeomcInit();
```

- `jeomcInit()`, a function call that is required to initiate any drawing behavior, contains all the startup functionality. `jeomcInit()` assesses the operating system that the program is being run on, calls the associated OS-appropriate libraries, and sets up the OpenGL window in which the image will be rendered.

3.5.2.6 jeomcRunAndSave()

```
void jeomcRunAndSave();
```

- `jeomcRunAndSave()` is a loop that is required to actually execute the shapes once rendered. All drawing functions queue an image into a buffer; `jeomcRunAndSave()` then reads the buffer and displays the contents in the window. While the window is running, the window contents are also automatically saved as a PNG file. At window close (i.e., when the user closes the window), execution of this loop terminates.

Note: For all of the above drawing functions, coordinates and numerical shape specifications can take any machine-interpretable value, but will only be rendered in the window, which ranges from [-1, 1] for both axes.

3.5.3 Hexcode support

For all drawing functions, color specifications are passed in via hex code string literals. These hex code strings are parsed in our backend to RGB float values. We support hex code string literals with or without a leading '`#`' character. Trailing whitespace will be ignored, but any white space leading or in between the string literal will affect parsing.

Our hex code parser will always read only the first 6 characters in the string literal, not including '#'. Whitespace-only or empty strings will default to 0.0, 0.0, 0.0 RGB values, and any invalid hex characters will cause the parser to terminate prematurely and save only the color information it has already parsed.

3.6 Program Structure (loop syntax, if/else statements)

JEo-MC supports if else conditionals as well as looping functionality, as below:

3.6.1 if and else statements

The if and else statements test a condition and based on its result, executes one part or the other of the program. In any given time, only one branch of the if/else statement should be executed.

```
bool wantCircle = true;
while (wantCircle)
  drawCircle(...);
else
  drawTriangle(...);
```

If desired, you may chain ifs together with elses to test for multiple conditions at one time.

3.6.2 while Statement

The while statement is a looping statement that evaluates the conditional surrounded in parentheses after it. If the conditional evaluates to true, the statements following will be evaluated. After the final statement within the loop has executed, the initial conditional is reevaluated, and this will go on until the conditional evaluates to false.

```
int count = 0;
while (count < 5)
  drawTriangle(...);
  count = count + 1;
```

3.6.3 for Statement

The for statement is another looping statement that allows for easier syntax when there is a need to keep track of a counter.
For simplicity, we will only allow a for loop if all three steps (initialize, test, and step) are filled out. This means you can't leave out for example the test part, to start in an infinite loop. If there is need for that, please consider the while statement.

```
int i;
for (i = 0; i < 5; i = i+1)
  drawTriangle(...);
```

3.6.4 break, continue, and return Statements

These keywords play a big role in looping logic in that they control the flow of the program. Here's an overview of what each keyword does:

`break`

A break statement will cause any loop to terminate immediately without reevaluating the conditional

`return`

Return functions similarly to break in that it causes a loop to terminate, but it simultaneously returns a value for the function and then quits the function

`continue`

A continue statement will allow a user to skip any statements following the current one, and immediately reevaluate the conditional at the current state.

3.6.5 Note on Loop Control

Please note that because our implementation of a for loop is the same as a while loop with the expressions shifted around, users must increment the counter explicitly before calling `continue` or risk infinite looping. This is a bit different from languages that implement the for loop differently, such as Java.

3.7 Exceptions

3.7.1 Keyword termination

Nothing may follow keywords like break, continue, and return

```
Fatal error: exception Failure("nothing may follow a break")
Fatal error: exception Failure("nothing may follow a continue")
Fatal error: exception Failure("nothing may follow a return")
```

3.7.2 Function arguments

Functions are verified to check whether number of args match, and all data types match

```
Fatal error: exception Failure("expecting 0 arguments in jeomcInit(x)")
Fatal error: exception Failure("illegal argument found float expected string
in x")
```

3.7.3 Binary Operators and Assignment

Assignment and binary operators check whether the two sides have the same defined type

```
Fatal error: exception Failure("illegal binary operator string == int in j == i")
Fatal error: exception Failure("illegal binary operator bool + int in d + a")
Fatal error: exception Failure("illegal binary operator float && int in -3.5 && 1")
Fatal error: exception Failure("illegal assignment int = bool in i = false")
```

3.7.4 Duplicate functions and variables

```
Fatal error: exception Failure("duplicate function bar")
Fatal error: exception Failure("duplicate formal a")
Fatal error: exception Failure("duplicate global b")
```

3.7.5 Unrecognized or undefined functions and variables

```
Fatal error: exception Failure("unrecognized function foo")
Fatal error: exception Failure("undeclared identifier j")
Fatal error: exception Failure("function print may not be defined")
```

3.7.6 General parsing errors

```
Fatal error: exception Stdlib.Parsing.Parse_error
```

4 Project Plan

4.1 Planning Process

Following initial formation of our group and language idea, our group initially met weekly to meet the earlier deliverables of the project, including the proposal and LRM. Later deliverables, including “Hello World,” compiler debugging, expansion of more complex features and developing test cases were completed on an individual basis. Especially in the later half of the semester, communication and collaboration took place primarily over virtual channels.

4.2 Project Timeline

1. **September 10-17:** Find a team to create a project idea with

2. **September 18-30:** Work on the team's project proposal
    a. Describe the language that you plan to implement.
    b. Explain what sorts of programs are meant to be written in your language
    c. Explain the parts of your language and what they do
    d. Include the source code for an interesting program in your language
3. **October 1-29:** Work on the Language Reference Manual and the Parser
4. **October 29-November 12**: Work on the "Hello World" demo specific to JEo-MC (generating an image)
5. **November 13-25:** Add JEo-MC features from LRM to our version of MicroC and implement drawing functions in GLFW and GLEW
6. **November 26-December 12:** Building out additional core features
7. **December 12-21:** Generating tests, working on final report and slides
8. **December 22:** Final Presentation

4.3 Development Environment

Libraries & Languages: C, OpenGL, GLSL, OCaml, Ocamlyacc, Ocamllex, LLVM
IDE: Visual Studio Code, IntelliJ
OS: macOS and Ubuntu
Version Control: Github

4.4 Programming Style Guide

The following specifications are not syntactically required for compilation, but are conventions that JEo-MC programmers should follow.

Overall structure
Above all, JEoMC code should be clean and readable. We recommend using lines of no longer than 80 characters (breaking long lines that might surpass this limit), keeping method declarations reasonably short by breaking complex functionality into several smaller functions, and avoiding unnecessarily long variable names. Comments should be descriptive and concise; where appropriate, block comments are preferred at the start of the file and inline comments should be used sparingly and concisely.

Global variables should be declared at the top of the function, even if they are not used until later in the program. Supporting functions should be declared right below variables, and the main method should be declared at the bottom.

Function declaration
Functions should be declared using camelCase. Function names should be short and descriptive, specifying the function's purpose without being overly verbose.

A good example of function declaration:

```
int foo(int a)
{
  return a;
}

OR

int foo(int a){
  return a;
}
```

Variable declaration

Variables should also be declared using camelCase. Variable names need not be overly long; for example, a simple character for an index variable is sufficient.

A good example of variable declaration:

```
float j;
```

Whitespace

Although we do not require whitespace, it is good practice to pad binary operators (i.e., =, +) with a space on either side. Between unary operators, note that there cannot be whitespace (this is an explicit requirement). Spacing should be consistent.

A good example of whitespace usage:

```
for (c = 0; c < 17; c = c + 1) { /*...*/
```

A bad example of whitespace usage:

```
drawRectangle(i, j+0.05 , i+0.05,j+0.05,   i+0.05,j  ,  i,j
,"#516091");
```

Tabs are suggested to clearly specify in-function code, where each subsequent scope depth increases the tab amount. Tabs should be equivalent to 4 spaces.

Additional lines of whitespace are suggested between logical separations in the code. These might include

- breaking up long functions into chunks that perform one purpose;

- separating method definitions from each other;
- splitting up large blocks of any type of code

Do **not** place a newline between individual global variable declarations; as a general rule, no single variable declaration should be spaced out from the others (i.e., place them in groups of at least 2).

Comments
Comments are encouraged, including e.g. block comments at the header of a file, or inline comments to delineate different parts of a more complex function call. Inline comments should be at the same level of indentation as the code they are commenting on.

An example of an appropriate header:

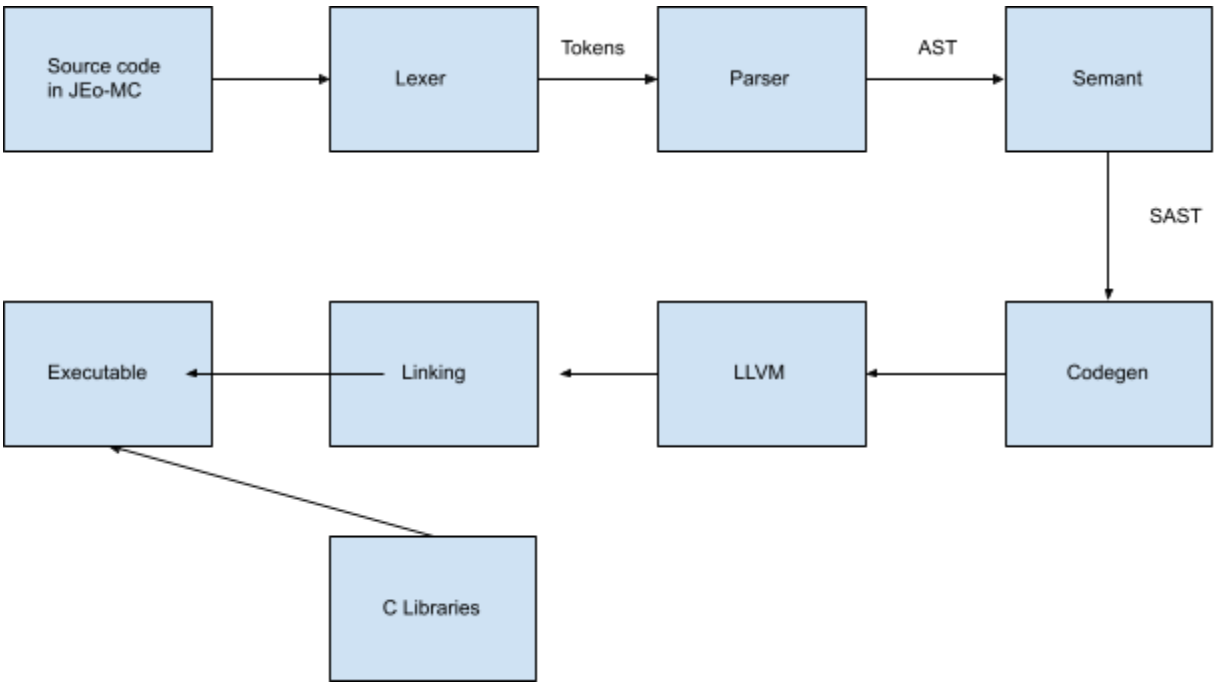
```
/*
    This function uses a while loop to generate
    a set of coordinates that will be used to draw
    a rectangle.
*/
```

4.5 Project Log

See Appendix 9.4 for the full log.

5 Architectural Design

5.1 Block Diagram



5.2 Description of Interfaces between Components

5.2.1 Source Code

This is the program written in JEo-MC that the user writes. The convention of our language requires files to end in ".j".

5.2.2 Lexer

The lexer, or the scanner (scanner.mll) translates ASCII characters in the JEo-MC source code and translates them into a stream of tokens and removes comments. Errors are thrown if illegal characters are found and lexed by the scanner.

5.2.3 Parser and AST

The lexed tokens are passed to the parser (parser.mly), which ensures that the program is syntactically correct. The parser converts the lexed tokens into an abstract syntax tree (AST)

with unique features to JEo-MC that were not part of the original program and help with semantic checking. If syntactic errors are found, errors by the parser will be thrown.

5.2.4 Semantic Checker

The AST is passed to the semantic checker (semant.ml). Semant.ml checks if the program is semantically correct. Next, semant.ml converts the AST into a semantically-checked abstract syntax tree (SAST).

5.2.5 Code Generation

The SAST is created to help with code generation (codegen.ml), which traverses through the SAST to create LLVM code.

5.2.6 LLVM

LLVM compiler, llc, converts an input .ll file into a .s assembly file
`llc -relocation-model=pic test.ll > test.s`

5.2.7 C Libraries - jeomc_draw.c

C libraries such as jeomc_draw.c are compiled into an object file jeomc_draw.o. This object will be linked when we compile the assembly file to produce the final executable. To compile source code into an object file (omitting linkers and other flags):
`cc -c jeomc_draw.c`

5.2.8 Linking

Object files such as jeomc_draw.c can be linked with the assembly file produced by LLVM to produce an executable as so (omitting linkers and other flags):
`cc -o test.exe test.s jeomc_draw.o`

5.3 Implementation Authors

5.3.1 Jeremy

- Linked initial C drawing function to OCaml compiler
- Set up docker in docker hub for project under lujeremy/jeomc
- Set up makefile to compile all dependencies, run tests and demo
- Set up version control on github, reviewed pull requests and resolved merge conflicts
- Modified testall.sh to compile tests with proper C flags for Linux; modified testall.sh and C imports to support both macOS & Linux
- Added break & continue keywords

- Created demo.sh, which runs all of our demos under ./demo
- Split drawing code in C into smaller, modular functions for translation into jeomc functions in OCaml (jeomcInit, jeomcRunAndSave, drawShape)
- Implemented drawRectangle(...)
- Implemented hex code parsing; add support for functions to support string parameters
- Adding ability to pass RGB information into vertex array itself before shader compilation
- Helped teammates troubleshoot issues with building our codebase on their machines
- Final report/presentation

5.3.2 Emma

- Modified compiler to recognize string primitives and String Literals in JEoMC programs
- Help modify compiler in general with help from homework 1
- Created “prints” function in compiler for JEoMC users to print strings
- Created tests to verify compiler was working correctly
- Had to troubleshoot LLVM problems with macOS
- Helped with the final report as my main priority over presentation

5.3.3 Melody

- Implemented drawTriangle(...), drawCircle(...), drawLine(...)
- Added functionality to save an image via saveImage()
- Modified testall.sh to compile tests with proper C flags for macOS
- Worked on an initial minimal compiler example
- Troubleshooting OpenGL installation on macOS, installation requirements, versioning
- Adding ability for colors to be modified via function parameters
- Wrote programs for the presentation slides and demo
- Final report/presentation

5.3.4 Connie

- Tried to fix linker issue due to M1 chip (Did not get fixed…)
- Added multiple tests
- Helped write final report/presentation

6 Test Plan

6.1 Testing Suite and Integration

Our testing suite consists of two parts, microc-tests and jeomc-tests. MicroC tests are taken directly from MicroC, to ensure that we are not breaking any of the existing capabilities of

MicroC. JEo-MC tests are tests written by ourselves to test any new features we have implemented on top, such as drawing functions, strings, and additional keywords.
Working tests begin with “test-”, in comparison to our purposefully failing tests beginning with “fail-”. Both successful and failed tests end with “.j”, as in JEoMC. The output files for the tests depend on if it passes or fails: ending with “.out” for successful tests and “.err” for failing tests. As long as JEoMC compiles, the testing script runs as well.

Tests can be run by this command: `$ make test`

See appendix for a full code listing of tests.

6.2 Unit and Integration Testing

Test cases were chosen to test out each different new functionality we introduced. For our drawing functions, because it is not possible to test whether a circle is actually rendered, we have made the decision to print out the vertices of the shapes being rendered, as well as the RGB values of the color passed in. In this way, we are able to test the output by comparing the printed output with what we expect.

Test cases for each feature was chosen carefully to test out how different edge cases would work. For example, for testing hexcode parsing, some edge cases would include:

- What if an empty string is passed in?
- What if the string consists of only whitespace?
- Will a leading ‘#’ affect parsing?
- What if invalid hex code characters are in the string?

In this way, we are able to make sure that we understand completely how our program will behave under different scenarios.

6.3 Test Automation

We have a testall.sh script to runs all tests in both jeomc-tests and microc-tests. This script takes a test file and runs it end to end to create the final output, then compares it with an expected output that we manually provide. If there is no difference, a tests is labeled as passing, while if there is a difference, a .diff file is generated and test is marked as failed. This script supports macOS and Linux operating systems, as it generates a set of unique linker flags for C compilation for each OS.

6.4 Test Authors

6.4.1 Jeremy

- fail-break.j
- fail-continue.j
- fail-draw-fun1.j
- fail-draw-fun2.j
- fail-draw-fun3.j
- fail-string.j
- fail-string-int-comparison.j
- test-break.j
- test-break2.j
- test-continue.j
- test-continue2.j
- test-draw1.j
- test-draw2.j
- test-empty-string.j
- test-for-add.j
- test-hex1.j
- test-hex2.j
- test-hex3.j
- test-hex4.j
- test-int.j
- test-int2.j
- test-string.j
- test-string2.j
- test-string-case.j
- loop-draw.j
- loop-draw2.j

6.4.2 Emma

- test-bool.j
- test-bool2.j
- test-bool3.j
- test-for-and.j
- test-hello2.j
- test-string3.j

6.4.3 Melody

- basic_circle1.j

- basic_circle2.j
- basic_tri_circle.j
- deathly.j
- snowman.j
- surprise.j

6.4.4 Connie

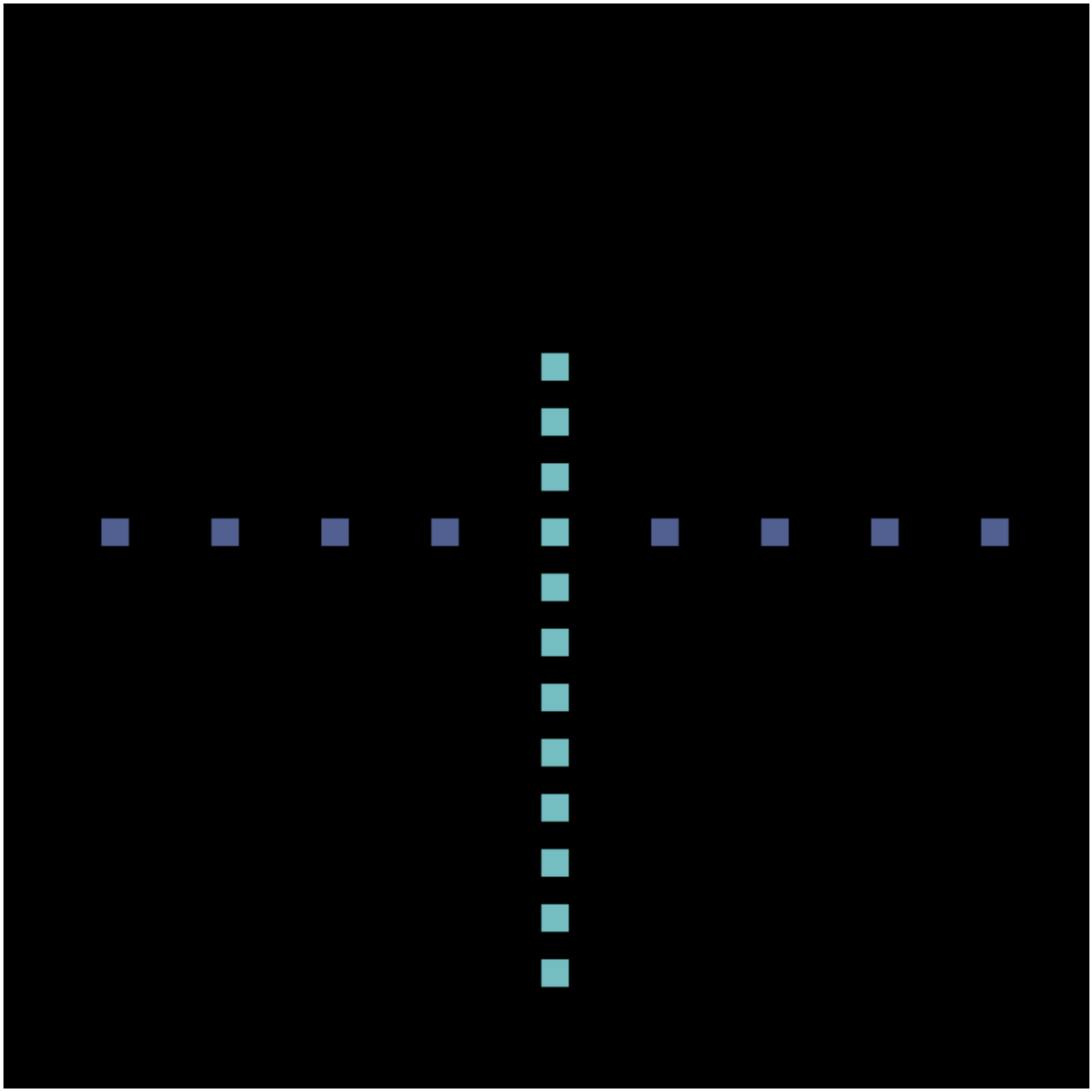
- test-arith4.j
- test-arith5.j
- test-for-div.j
- test-for-minus.j
- test-for-mul.j

6.5 Examples of Source Language Programs

Below is a representative JEoMC program of significant complexity. On the drawing side, this program demonstrates the ability to draw multiple shapes in a single window and switching colors between them. On the syntactic side, this program illustrates the `break` and `continue` functionality by halting and continuing execution at specified break/continue points. In addition, the program demonstrates usage of negative numbers, simple arithmetic, boolean logic, and appropriate commenting.

```
int main(){
	float i;
	float j;
	int c;

	jeomcInit();

	i = -0.8;
	j = 0.0;
	c = 0;
	for (c = 0; c < 17; c = c + 1) {
	/* use continue to skip all odd positions */
		if (c == 1 || c == 3 || c == 5 || c == 7 || c == 9 || c == 11 || c == 13 {
			c = c + 1;
			i = i + 0.1;
			continue;
		}
		drawRectangle(i, j+0.05, i+0.05, j+0.05, i+0.05, j, i, j,"#516091");
		i = i + 0.1;
	}

	i = 0.0;
	j = -0.8;
	c = 0;
```

```
        for (c = 0; c < 15; c = c + 1) {
        /* after drawing 12 rectangles, break */
                if (c == 12) {
                        break;
                }
                drawRectangle(i, j+0.05, i+0.05, j+0.05, i+0.05,j, i, j,"#74bec1");
                j = j + 0.1;
        }

        jeomcRunAndSave();

        return 0;
}
```



Below is an example drawing program that was used as a live demo. The following code generates the accompanying image.

```
int main(){

  jeomcInit();

  drawRectangle(-1.0,-1.0,-1.0,1.0,1.0,1.0,1.0,-1.0,"#A1CDCF");
  drawRectangle(-1.0,-1.0,-1.0,0.0,1.0,0.0,1.0,-1.0,"#FCFEFD");

  drawTriangle(0.0,-1.0,0.5,0.7,2.0,-1.0,"#C2CECF");
  drawCircle(0.5,0.3,0.8,"#C4C3B1");
  drawCircle(0.5,0.2,0.6,"#282923");
  drawCircle(0.5,0.05,0.5,"#FBE2CA");
  drawTriangle(0.30,-0.2,0.5,-0.30,0.7,-0.2,"#FDEDDE");

  drawTriangle(0.2,0.12,0.3,0.17,0.4,0.12,"#210F00");
  drawTriangle(0.6,0.12,0.7,0.17,0.8,0.12,"#210F00");

  drawCircle(-0.1,-0.5,0.25,"#FBE2CA");

  drawRectangle(0.0,-0.4,-0.1,-0.7,1.0,-1.0,1.0,-0.6,"#CED5D5");

  drawCircle(-0.4,-0.4,0.3,"#DB8200");
  drawRectangle(-0.75,-0.50,-0.75,-0.35,-0.05,-0.35,-0.05,-0.5,"#F3C519");

  drawLine(0.4,0.6,0.45,0.7,"FCFF53");
  drawLine(0.45,0.7,0.5,0.65,"FCFF53");
  drawLine(0.5,0.65,0.55,0.7,"FCFF53");
  drawLine(0.55,0.7,0.6,0.6,"FCFF53");



  jeomcRunAndSave();

}
```

7 Example Test Programs

See appendix for a full code listing of demo files used on 12/22/2021.

8 Lessons Learned

8.1 Jeremy

One important lesson I learned is to spend more time exploring different libraries and researching how they are used. While in the end we managed to make OpenGL/glfw work, we didn't need some of the more advanced features it offers. The trade-off for the advanced drawing capabilities of OpenGL was that it was difficult to use and hard to understand. The road to getting drawings in a window could have been much easier if we chose a simpler library (e.g. SDL2). We found these other libraries a bit too late in the process however and continued on with OpenGL.

It is also important to align dependency versions between the team, as we encountered many roadblocks because we compiled our code on different machines and operating systems

with mismatching OpenGL versions. Docker would have solved this issue if not for the fact that opening a drawing window there is difficult.

One final word of advice I would give to future groups is to identify and escalate team issues early on. Have patience for someone not understanding something, but have no tolerance for someone who does not try. After several attempts of trying to engage members who are not putting in effort, give up. Don't beat a dead horse or wait until it is too late. Listen carefully to every single one of Professor Edwards' advice on the first day, especially "if someone is really underperforming, dump his ass."

8.2 Emma

I found it rewarding and challenging to communicate throughout the process of developing JEo-MC. I have trouble reaching out and asking for help when I assume everyone around me knows what they are doing. As a Junior in a class filled with graduate students, I was extremely intimidated. However, I overcame my intimidation and used Hao's Office Hours to my advantage as well as communicating with my classmates about things I did not understand originally. I have learned that everyone is willing to help, and no one is an expert or above anyone else. Communicating began from being consistently in person to online as the semester progressed. In a hybrid environment, communication is difficult but necessary in order to be successful. Working on the same concepts at once virtually did have some setbacks. For example, a group member asked for help with the drawing function. As I was getting myself familiarized with it, the group member was already finished and did not need my help any longer. I felt like this would happen often, and I kept feeling more and more guilty about not being incredibly familiar with the concepts learned in this course. However, I have learned to become a more flexible and adaptable learner and teammate throughout this experience. I now feel confident in what I created during this project and would do it all over again if I could.

I also learned how tedious all of our devices are. We ran into many roadblocks when compiling the same version of code across all of our machines that would produce different results somehow. It was frustrating and I still do not understand why our computers reacted differently, but it allowed me to practice being patient with things not in my control. Usually after a google search, we were able to dissect, discover, and solve the problem.

If I were to give advice to future teams, I would beg them to begin brainstorming immediately. I know the end of the semester seems months away, but it will come in an instant. Making sure all team members have the correct software and understanding of the project at hand is key as well. Also, create a team with people who are friendly and have an open mind because you are stuck with them for a while!x

8.3 Melody

Developing a project of this size was a great lesson in time management, effective collaboration, and version control. Learning to prioritize effectively - whether to implement a

feature, whether to leave it out, if the effort required to add or implement something would be worth the eventual benefit - was something that I had to learn along the way. Another big takeaway for me was the importance of learning to work with individuals with diverse work styles and how to manage expectations and cooperate seamlessly despite these differences. Ultimately, it was quite rewarding to see the visual fruits of my labor rendered in colorful shapes and lines.

8.4 Connie

Well, for starters, learning about compilers. I never knew it was so complicated under the hood and seeing our language come to life was rewarding. In addition to that, the drawing libraries were really cool and I never knew that C could make shapes and display them.
As for other aspects…Communication is definitely the most important thing when it comes to group projects. Communicate and clarify tasks with teammates! I found myself lost multiple times during the project and had to catch up individually because I didn't ask enough questions when it was first getting implemented. People are nice so just asking does wonders. I also felt that taking time to understand the code myself would really help in the long run.
Being more initiative = Being able to catch up quicker or help the team manage the timeline better would have definitely helped us each do a lot better.

For future teams, I would just say this. Choose your teammates wisely. Find out their priorities, schedules, how often they can meet, where they live for meeting in person etc… Be friends with them. Assign roles at the beginning of the semester and help each other out rather than just leaving it as a "anyone can do anything" situation. It doesn't really work. Let them ask you questions and vice versa. AND START EARLIER. The end of the semester is extremely hectic, better have things done earlier than rushing and sleeping at 3, 4, 5am at night.

9 Appendix

9.1 Compiler

9.1.1 parser.mly

```
/* Ocamlyacc parser for JEo-MC */

%{
open Ast
%}

%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA PLUS MINUS TIMES DIVIDE
```

```
ASSIGN
%token NOT EQ NEQ LT LEQ GT GEQ AND OR
%token RETURN IF ELSE FOR WHILE INT BOOL FLOAT VOID
%token CONTINUE BREAK
%token <int> LITERAL
%token <bool> BLIT
%token <string> ID FLIT
%token EOF

%start program
%type <Ast.program> program

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE
%right NOT

%%

program:
  decls EOF { $1 }

decls:
   /* nothing */ { ([], [])               }
 | decls vdecl { (($2 :: fst $1), snd $1) }
 | decls fdecl { (fst $1, ($2 :: snd $1)) }

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

formals_opt:
```

```
    /* nothing */ { [] }
  | formal_list   { $1 }

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

typ:
    INT   { Int   }
  | BOOL  { Bool  }
  | FLOAT { Float }
  | VOID  { Void  }

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

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

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

stmt:
    expr SEMI                               { Expr $1               }
  | RETURN expr_opt SEMI                    { Return $2             }
  | LBRACE stmt_list RBRACE                 { Block(List.rev $2)    }
  | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
  | IF LPAREN expr RPAREN stmt ELSE stmt    { If($3, $5, $7)        }
  | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
                                            { For($3, $5, $7, $9)   }
  | WHILE LPAREN expr RPAREN stmt           { While($3, $5)         }
  | BREAK SEMI                              { Break }
  | CONTINUE SEMI                           { Continue }

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

expr:
    LITERAL          { Literal($1)            }
```

```
  | FLIT         { Fliteral($1)           }
  | BLIT             { BoolLit($1)            }
  | ID               { Id($1)                 }
  | expr PLUS   expr { Binop($1, Add,   $3)   }
  | expr MINUS  expr { Binop($1, Sub,   $3)   }
  | expr TIMES  expr { Binop($1, Mult,  $3)   }
  | expr DIVIDE expr { Binop($1, Div,   $3)   }
  | expr EQ     expr { Binop($1, Equal, $3)   }
  | expr NEQ    expr { Binop($1, Neq,   $3)   }
  | expr LT     expr { Binop($1, Less,  $3)   }
  | expr LEQ    expr { Binop($1, Leq,   $3)   }
  | expr GT     expr { Binop($1, Greater, $3) }
  | expr GEQ    expr { Binop($1, Geq,   $3)   }
  | expr AND    expr { Binop($1, And,   $3)   }
  | expr OR     expr { Binop($1, Or,    $3)   }
  | MINUS expr %prec NOT { Unop(Neg, $2)      }
  | NOT expr         { Unop(Not, $2)          }
  | ID ASSIGN expr   { Assign($1, $3)         }
  | ID LPAREN args_opt RPAREN { Call($1, $3)  }
  | LPAREN expr RPAREN { $2                   }

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

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

9.1.2 scanner.mll

```
(* Ocamllex scanner for JEo-MC *)

{ open Parser }

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

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

```
| ')'      { RPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
| ';'      { SEMI }
| ','      { COMMA }
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '='      { ASSIGN }
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
| ">"      { GT }
| ">="     { GEQ }
| "&&"     { AND }
| "||"     { OR }
| "!"      { NOT }
| "if"     { IF }
| "else"   { ELSE }
| "for"    { FOR }
| "while"  { WHILE }
| "return" { RETURN }
| "break"  { BREAK }
| "continue" { CONTINUE }
| "int"    { INT }
| "bool"   { BOOL }
| "float"  { FLOAT }
| "void"   { VOID }
| "true"   { BLIT(true)  }
| "false"  { BLIT(false) }
| digits as lxm { LITERAL(int_of_string lxm) }
| digits '.'  digit* ( ['e' 'E'] ['+' '-']? digits )? as lxm { FLIT(lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*     as lxm { ID(lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

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

9.1.3 ast.ml

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

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

type uop = Neg | Not

type typ = Int | String | Bool | Float | Void

type bind = typ * string

type expr =
    Literal of int
  | Slit of string
  | Flit of string
  | BoolLit of bool
  | Id of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Assign of string * expr
  | Call of string * expr list
  | Noexpr

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

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

```
type program = bind list * func_decl list

(* Pretty-printing functions *)

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

let string_of_uop = function
    Neg -> "-"
  | Not -> "!"

let rec string_of_expr = function
    Literal(l) -> string_of_int l
  | Slit(l) -> "\"" ^ l ^ "\""
  | Flit(l) -> l
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | Id(s) -> s
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
  | Assign(v, e) -> v ^ " = " ^ string_of_expr e
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | Noexpr -> ""

let rec string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
```

```
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
  | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^
string_of_stmt s
  | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
      string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
  | For(e1, e2, e3, s) ->
      "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ; " ^
      string_of_expr e3  ^ ") " ^ string_of_stmt s
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s
  | Break -> "break;\n"
  | Continue -> "continue;\n"

let string_of_typ = function
    Int -> "int"
  | String -> "string"
  | Bool -> "bool"
  | Float -> "float"
  | Void -> "void"

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

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

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

9.1.4 sast.ml

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

open Ast
```

```
type sexpr = typ * sx
and sx =
    SLiteral of int
  | SSlit of string
  | SFlit of string
  | SBoolLit of bool
  | SId of string
  | SBinop of sexpr * op * sexpr
  | SUnop of uop * sexpr
  | SAssign of string * sexpr
  | SCall of string * sexpr list
  | SNoexpr

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

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

type sprogram = bind list * sfunc_decl list

(* Pretty-printing functions *)

let rec string_of_sexpr (t, e) =
  "(" ^ string_of_typ t ^ " : " ^ (match e with
    SLiteral(l) -> string_of_int l
  | SBoolLit(true) -> "true"
  | SBoolLit(false) -> "false"
  | SSlit(l) -> "\"" ^ l ^ "\""
  | SFlit(l) -> l
```

```
  | SId(s) -> s
  | SBinop(e1, o, e2) ->
      string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_sexpr e2
  | SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
  | SAssign(v, e) -> v ^ " = " ^ string_of_sexpr e
  | SCall(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^ ")"
  | SNoexpr -> ""
                          ) ^ ")"

let rec string_of_sstmt = function
    SBlock(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_sstmt stmts) ^ "}\n"
  | SExpr(expr) -> string_of_sexpr expr ^ ";\n";
  | SReturn(expr) -> "return " ^ string_of_sexpr expr ^ ";\n";
  | SIf(e, s, SBlock([])) ->
      "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s
  | SIf(e, s1, s2) ->  "if (" ^ string_of_sexpr e ^ ")\n" ^
      string_of_sstmt s1 ^ "else\n" ^ string_of_sstmt s2
  | SFor(e1, e2, e3, s) ->
      "for (" ^ string_of_sexpr e1  ^ " ; " ^ string_of_sexpr e2 ^ " ; " ^
      string_of_sexpr e3  ^ ") " ^ string_of_sstmt s
  | SWhile(e, s) -> "while (" ^ string_of_sexpr e ^ ") " ^ string_of_sstmt
s
  | SBreak -> "break;\n"
  | SContinue -> "continue;\n"

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

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

9.1.5 semant.ml

```
(* Semantic checking for the JEo-MC compiler *)
```

```
open Ast
open Sast

module StringMap = Map.Make(String)

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

   Check each global variable, then check each function *)

let check (globals, functions) =

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

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

  check_binds "global" globals;

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

  (* Collect function declarations for built-in functions: no bodies *)
  let built_in_decls =
    let add_bind map (name, ty) = StringMap.add name {
      typ = Void;
      fname = name;
      formals = ty;
      locals = []; body = [] } map
    in List.fold_left add_bind StringMap.empty [  ("print", [(Int, "x")]);
                                                  ("printb", [(Bool,
"x")]);
```

```
                                                    ("printf", [(Float,
"x")]);
                                                    ("prints", [(String,
"x")]);
                                                    ("drawTriangle", [(Float,
"x1"); (Float, "y1"); (Float, "x2"); (Float, "y2"); (Float, "x3"); (Float,
"y3"); (String, "c");]);
                                                    ("drawCircle", [(Float,
"x"); (Float, "y"); (Float, "radius"); (String, "c");]);
                                                    ("drawRectangle",
[(Float, "x"); (Float, "y"); (Float, "w"); (Float, "h"); (String, "c")]);
                                                    ("drawLine", [(Float,
"x1"); (Float, "y1"); (Float, "x2"); (Float, "y2"); (String, "c")]);
                                                    ("jeomcInit", [ ]);
                                                    ("jeomcRunAndSave", [ ])
                                                    ]
  in

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

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

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

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

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

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

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

    (* Return a variable from our local symbol table *)
    let type_of_identifier s =
      try StringMap.find s symbols
      with Not_found -> raise (Failure ("undeclared identifier " ^ s))
    in

    (* Return a semantically-checked expression, i.e., with a type *)
    let rec expr = function
        Literal  l -> (Int, SLiteral l)
      | Slit l -> (String, SSlit l)
      | Flit l -> (Float, SFlit l)
      | BoolLit l  -> (Bool, SBoolLit l)
      | Noexpr     -> (Void, SNoexpr)
      | Id s       -> (type_of_identifier s, SId s)
      | Assign(var, e) as ex ->
          let lt = type_of_identifier var
          and (rt, e') = expr e in
          let err = "illegal assignment " ^ string_of_typ lt ^ " = " ^
            string_of_typ rt ^ " in " ^ string_of_expr ex
          in (check_assign lt rt err, SAssign(var, (rt, e')))
      | Unop(op, e) as ex ->
          let (t, e') = expr e in
          let ty = match op with
```

```
            Neg when t = Int || t = Float -> t
          | Not when t = Bool -> Bool
          | _ -> raise (Failure ("illegal unary operator " ^
                                 string_of_uop op ^ string_of_typ t ^
                                 " in " ^ string_of_expr ex))
          in (ty, SUnop(op, (t, e')))
      | Binop(e1, op, e2) as e ->
          let (t1, e1') = expr e1
          and (t2, e2') = expr e2 in
          (* All binary operators require operands of the same type *)
          let same = t1 = t2 in
          (* Determine expression type based on operator and operand types
*)
          let ty = match op with
            Add | Sub | Mult | Div when same && t1 = Int   -> Int
          | Add | Sub | Mult | Div when same && t1 = Float -> Float
          | Equal | Neq            when same               -> Bool
          | Less | Leq | Greater | Geq
                     when same && (t1 = Int || t1 = Float) -> Bool
          | And | Or when same && t1 = Bool -> Bool
          | _ -> raise (
            Failure ("illegal binary operator " ^
                       string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
                       string_of_typ t2 ^ " in " ^ string_of_expr e))
          in (ty, SBinop((t1, e1'), op, (t2, e2')))
      | Call(fname, args) as call ->
          let fd = find_func fname in
          let param_length = List.length fd.formals in
          if List.length args != param_length then
            raise (Failure ("expecting " ^ string_of_int param_length ^
                            " arguments in " ^ string_of_expr call))
          else let check_call (ft, _) e =
            let (et, e') = expr e in
            let err = "illegal argument found " ^ string_of_typ et ^
              " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr e
            in (check_assign ft et err, e')
          in
          let args' = List.map2 check_call fd.formals args
          in (fd.typ, SCall(fname, args'))
    in

    let check_bool_expr e =
```

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

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

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

    in (* body of check_function *)
    { styp = func.typ;
```

```
      sfname = func.fname;
      sformals = func.formals;
      slocals  = func.locals;
      sbody = match check_stmt (Block func.body) with
      SBlock(sl) -> sl
      | _ -> raise (Failure ("internal error: block didn't become a
block?"))
    }
  in (globals, List.map check_function functions)
```

9.1.6 codegen.ml

```
(* Code generation: translate takes a semantically checked AST and
produces LLVM IR

LLVM tutorial: Make sure to read the OCaml version of the tutorial

http://llvm.org/docs/tutorial/index.html

Detailed documentation on the OCaml LLVM library:

http://llvm.moe/
http://llvm.moe/ocaml/

*)

module L = Llvm
module A = Ast
open Sast

module StringMap = Map.Make(String)

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

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

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

```
  let i32_t      = L.i32_type    context
  and i8_t       = L.i8_type     context
  and i1_t       = L.i1_type     context
  and string_t   = L.pointer_type (L.i8_type context)
  and float_t    = L.double_type context
  and void_t     = L.void_type   context
  and cnull      = L.const_null (L.i32_type context) in

  (* Return the LLVM type for a JEo-MC type *)
  let ltype_of_typ = function
      A.Int   -> i32_t
    | A.String -> string_t
    | A.Bool  -> i1_t
    | A.Float -> float_t
    | A.Void  -> void_t
  in

  let break_block = ref (cnull) in
  let continue_block = ref (cnull) in

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

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

  let drawTriangle_t : L.lltype =
      L.function_type i32_t [| float_t; float_t; float_t; float_t; float_t;
float_t; string_t |] in
  let drawTriangle_func : L.llvalue =
      L.declare_function "drawTriangle" drawTriangle_t the_module in

  let drawCircle_t : L.lltype =
      L.function_type i32_t [| float_t; float_t; float_t; string_t |] in
```

```
  let drawCircle_func : L.llvalue =
      L.declare_function "drawCircle" drawCircle_t the_module in

  let drawRectangle_t : L.lltype =
      L.function_type i32_t [| float_t; float_t; float_t; float_t; string_t
|] in
  let drawRectangle_func : L.llvalue =
      L.declare_function "drawRectangle" drawRectangle_t the_module in

  let drawLine_t : L.lltype =
      L.function_type i32_t [| float_t; float_t; float_t; float_t; string_t
|] in
  let drawLine_func : L.llvalue =
      L.declare_function "drawLine" drawLine_t the_module in

  let jeomcInit_t : L.lltype =
      L.function_type i32_t [| |] in
  let jeomcInit_func : L.llvalue =
      L.declare_function "jeomcInit" jeomcInit_t the_module in

  let jeomcRunAndSave_t : L.lltype =
      L.function_type i32_t [| |] in
  let jeomcRunAndSave_func : L.llvalue =
      L.declare_function "jeomcRunAndSave" jeomcRunAndSave_t the_module in

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

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

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

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

      (* Allocate space for any locally declared variables and add the
       * resulting registers to our map *)
      and add_local m (t, n) =
      let local_var = L.build_alloca (ltype_of_typ t) n builder
      in StringMap.add n local_var m
      in

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

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

    (* Construct code for an expression; return its value *)
    let rec expr builder ((_, e) : sexpr) = match e with
        SLiteral i  -> L.const_int i32_t i
      | SBoolLit b  -> L.const_int i1_t (if b then 1 else 0)
      | SFlit l -> L.const_float_of_string float_t l
      | SSlit s     -> L.build_global_stringptr (s ^ "\x00") "str" builder
      | SNoexpr     -> L.const_int i32_t 0
```

```
      | SId s       -> L.build_load (lookup s) s builder
      | SAssign (s, e) -> let e' = expr builder e in
                          ignore(L.build_store e' (lookup s) builder); e'
      | SBinop ((A.Float,_ ) as e1, op, e2) ->
        let e1' = expr builder e1
        and e2' = expr builder e2 in
        (match op with
          A.Add     -> L.build_fadd
        | A.Sub     -> L.build_fsub
        | A.Mult    -> L.build_fmul
        | A.Div     -> L.build_fdiv
        | A.Equal   -> L.build_fcmp L.Fcmp.Oeq
        | A.Neq     -> L.build_fcmp L.Fcmp.One
        | A.Less    -> L.build_fcmp L.Fcmp.Olt
        | A.Leq     -> L.build_fcmp L.Fcmp.Ole
        | A.Greater -> L.build_fcmp L.Fcmp.Ogt
        | A.Geq     -> L.build_fcmp L.Fcmp.Oge
        | A.And | A.Or ->
            raise (Failure "internal error: semant should have rejected
and/or on float")
        ) e1' e2' "tmp" builder
      | SBinop (e1, op, e2) ->
        let e1' = expr builder e1
        and e2' = expr builder e2 in
        (match op with
          A.Add     -> L.build_add
        | A.Sub     -> L.build_sub
        | A.Mult    -> L.build_mul
    | A.Div     -> L.build_sdiv
        | A.And     -> L.build_and
        | A.Or      -> L.build_or
        | A.Equal   -> L.build_icmp L.Icmp.Eq
        | A.Neq     -> L.build_icmp L.Icmp.Ne
        | A.Less    -> L.build_icmp L.Icmp.Slt
        | A.Leq     -> L.build_icmp L.Icmp.Sle
        | A.Greater -> L.build_icmp L.Icmp.Sgt
        | A.Geq     -> L.build_icmp L.Icmp.Sge
        ) e1' e2' "tmp" builder
      | SUnop(op, ((t, _) as e)) ->
          let e' = expr builder e in
        (match op with
          A.Neg when t = A.Float -> L.build_fneg
```

```
        | A.Neg                  -> L.build_neg
          | A.Not                  -> L.build_not) e' "tmp" builder
      | SCall ("print", [e]) | SCall ("printb", [e]) ->
        L.build_call printf_func [| int_format_str ; (expr builder e) |]
          "printf" builder
      | SCall ("prints", [e]) ->
    L.build_call printf_func [| str_format_str ; (expr builder e) |]
"printf" builder
      | SCall ("jeomcInit", []) ->
    L.build_call jeomcInit_func [| |] "jeomcInit" builder
      | SCall ("jeomcRunAndSave", []) ->
    L.build_call jeomcRunAndSave_func [| |] "jeomcRunAndSave" builder
      | SCall ("drawTriangle", [x1;y1;x2;y2;x3;y3;c]) ->
    L.build_call drawTriangle_func [| (expr builder x1); (expr builder y1);
(expr builder x2); (expr builder y2); (expr builder x3); (expr builder y3);
(expr builder c); |] "drawTriangle" builder
      | SCall ("drawCircle", [x;y;radius;c]) ->
    L.build_call drawCircle_func [| (expr builder x); (expr builder y);
(expr builder radius); (expr builder c); |] "drawCircle" builder
      | SCall ("drawRectangle", [x;y;w;h;c]) ->
    L.build_call drawRectangle_func [| (expr builder x); (expr builder y);
(expr builder w); (expr builder h); (expr builder c); |] "drawRectangle"
builder
      | SCall ("drawLine", [x1;y1;x2;y2;c]) ->
    L.build_call drawLine_func [| (expr builder x1); (expr builder y1);
(expr builder x2); (expr builder y2); (expr builder c); |] "drawLine"
builder
      | SCall ("printf", [e]) ->
        L.build_call printf_func [| float_format_str ; (expr builder e) |]
          "printf" builder
      | SCall (f, args) ->
         let (fdef, fdecl) = StringMap.find f function_decls in
       let llargs = List.rev (List.map (expr builder) (List.rev args)) in
       let result = (match fdecl.styp with
                        A.Void -> ""
                      | _ -> f ^ "_result") in
         L.build_call fdef (Array.of_list llargs) result builder
    in

    (* LLVM insists each basic block end with exactly one "terminator"
       instruction that transfers control.  This function runs "instr
builder"
```

```
       if the current block does not already have a terminator.  Used,
       e.g., to handle the "fall off the end of the function" case. *)
    let add_terminal builder instr =
      match L.block_terminator (L.insertion_block builder) with
      Some _ -> ()
      | None -> ignore (instr builder) in



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

    let rec stmt builder = function
      SBlock sl -> List.fold_left stmt builder sl
      | SExpr e -> ignore(expr builder e); builder
      | SBreak    -> ignore(L.build_br (L.block_of_value !break_block)
builder); builder
      | SContinue -> ignore(L.build_br (L.block_of_value !continue_block)
builder); builder
      | SReturn e -> ignore(match fdecl.styp with
                                  (* Special "return nothing" instr *)
                                  A.Void -> L.build_ret_void builder
                                  (* Build return statement *)
                                | _ -> L.build_ret (expr builder e) builder
);
                         builder
      | SIf (predicate, then_stmt, else_stmt) ->
         let bool_val = expr builder predicate in
         let merge_bb = L.append_block context "merge" the_function in
               let build_br_merge = L.build_br merge_bb in (* partial
function *)

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

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

         ignore(L.build_cond_br bool_val then_bb else_bb builder);
```

```
         L.builder_at_end context merge_bb

      | SWhile (predicate, body) ->
        let pred_bb = L.append_block context "while" the_function in
        let body_bb = L.append_block context "while_body" the_function in
        let merge_bb = L.append_block context "merge" the_function in
        let _ = L.build_br pred_bb builder in
        let _ = break_block := L.value_of_block merge_bb in
        let _ = continue_block := L.value_of_block pred_bb in
        let pred_builder = L.builder_at_end context pred_bb in
        let bool_val = expr pred_builder predicate in
        let _ = L.build_cond_br bool_val body_bb merge_bb pred_builder in

        add_terminal (stmt (L.builder_at_end context body_bb) body)
(L.build_br pred_bb);
        L.builder_at_end context merge_bb
          (* Implement for loops as while loops *)
      | SFor (e1, e2, e3, body) -> stmt builder
          ( SBlock [SExpr e1 ; SWhile (e2, SBlock [body ; SExpr e3]) ] )
        in

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

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

  List.iter build_function_body functions;
  the_module
```

9.2 Linker C program

9.2.1 jeomc_draw.c

```
#define STB_IMAGE_WRITE_IMPLEMENTATION
#include <GL/glew.h>
#include <GLFW/glfw3.h>
```

```c
#include "stb_image_write.h"
#include <stdio.h>
#define _USE_MATH_DEFINES
#include <math.h>

/* MacOS imports */
#if __has_include(<OpenGL/gl.h>)
#include <OpenGL/gl.h>
#endif
#if __has_include(<GLUT/glut.h>)
#include <GLUT/glut.h>
#endif

/* Linux imports */
#if __has_include(<GL/gl.h>)
#include <GL/gl.h>
#endif
#if __has_include(<GL/glut.h>)
#include <GL/glut.h>
#endif

/* Keep window, vao array, and an index as global variables */
GLFWwindow* window;


/* Initialize glfw, creating the window to the global var */
void jeomcInit() {

   /* Initialize the library */
   if (!glfwInit())
       return;

#ifdef __APPLE__
   glfwWindowHint (GLFW_CONTEXT_VERSION_MAJOR,3);
   glfwWindowHint (GLFW_CONTEXT_VERSION_MINOR,2);
   glfwWindowHint (GLFW_OPENGL_FORWARD_COMPAT,GL_TRUE);
   glfwWindowHint (GLFW_OPENGL_PROFILE,GLFW_OPENGL_CORE_PROFILE);
#endif

   /* Create a windowed mode window and its OpenGL context */
   window = glfwCreateWindow(640, 640, "JEoMC", NULL, NULL);
   if (!window)
```

```
    {
        glfwTerminate();
        return;
    }

    /* Make the window's context current */
    glfwMakeContextCurrent(window);
    glClearColor( 0.1f, 0.1f, 0.1f, 0.0f );

    /* Enable transparency */
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glEnable(GL_COLOR_MATERIAL);

    glewExperimental = GL_TRUE;
    glewInit();
    return;
}

void compileShaders() {
    const char* vertexSource =
      "#version 330\n"
      "layout (location = 0) in vec3 vp;    // Position in attribute
location 0\n"
      "layout (location = 1) in vec3 vc;  // Color in attribute location
1\n"
      "out vec3 theColor;                 // output a color to the fragment
shader\n"
      "void main() {"
      "    gl_Position = vec4(vp,1.0);"
      "    theColor = vc;\n"
      "}";

    const char* fragmentSource =
      "#version 330\n"
      "in vec3 theColor;      // Color value came from the vertex shader
(smoothed) \n"
      "out vec4 FragColor;    // Color that will be used for the fragment\n"
      "void main() {"
      "    FragColor = vec4(theColor, 1.0f);"
      "}";
```

```
   GLuint vs = glCreateShader(GL_VERTEX_SHADER);
   glShaderSource(vs,1,&vertexSource,NULL);
   glCompileShader(vs);
   GLuint fs = glCreateShader(GL_FRAGMENT_SHADER);
   glShaderSource(fs,1,&fragmentSource,NULL);
   glCompileShader(fs);

   GLuint shader_programme = glCreateProgram();
   glAttachShader(shader_programme,fs);
   glAttachShader(shader_programme,vs);
   glLinkProgram(shader_programme);
   glUseProgram(shader_programme);

   return;
}

void bindVertexBufferAndArray() {
   GLuint vbo, vao;
   glGenVertexArrays(1, &vao);
   glGenBuffers(1, &vbo);

   glBindVertexArray(vao);
   glBindBuffer(GL_ARRAY_BUFFER, vbo);

   glVertexAttribPointer(0, 3, GL_DOUBLE, GL_FALSE, 6*sizeof(double),
(void*)0);
   glEnableVertexAttribArray(0);
   glVertexAttribPointer(1, 3, GL_DOUBLE, GL_FALSE, 6*sizeof(double),
(void*)(3*sizeof(double)));
   glEnableVertexAttribArray(1);
   return;
}

void bindElementBuffer() {
   GLuint ebo;
   glGenBuffers(1, &ebo);
   glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, ebo);
}

void drawTriangle(double x1, double y1, double x2, double y2, double x3,
double y3, char hex[]) {
   if (hex[0] == '#')
```

```
    hex++;
  int num = (int) strtol(hex, NULL, 16);
  double r = ((num >> 16) & 0xFF) / 255.0;
  double g = ((num >> 8) & 0xFF) / 255.0;
  double b = (num & 0xFF) / 255.0;

  double points[] = {
    x1, y1, 0.0, r, g, b,
    x2, y2, 0.0, r, g, b,
    x3, y3, 0.0, r, g, b
  };

  printf("Drawing triangle:\n");
  for (int i = 0; i < sizeof(points) / sizeof(double); i+= 6) {
    printf("Vertex: (%lf, %lf, %lf) RGB: (%lf, %lf, %lf)\n", points[i],
points[i+1],points[i+2],points[i+3],points[i+4],points[i+5]);
  }

  printf("\n");

  // bind vao and vbo, buffer points
  bindVertexBufferAndArray();
  glBufferData(GL_ARRAY_BUFFER, sizeof(points), points, GL_STATIC_DRAW);

  // compile shaders
  compileShaders();

  // swap buffers, draw, then swap back
  glfwSwapBuffers(window);
  glDrawArrays(GL_TRIANGLES, 0, 3);
  glfwSwapBuffers(window);

  return;
}

void drawCircle(double centerx, double centery,double rad, char hex[]){
  if (hex[0] == '#')
    hex++;
  int num = (int) strtol(hex, NULL, 16);
  double r = ((num >> 16) & 0xFF) / 255.0;
  double g = ((num >> 8) & 0xFF) / 255.0;
  double b = (num & 0xFF) / 255.0;
```

```
   int theta = 0;
   double x;
   double y;
   double z = 0.0;
   int numPoints = 30;
   int currentSize = 0;
   double points[numPoints*6];

   while (theta<360){
       x = (double) rad * cos(theta * M_PI / 180.0);
       y = (double) rad * sin(theta * M_PI / 180.0);

       points[currentSize++] = x + centerx;
       points[currentSize++] = y + centery;
       points[currentSize++] = z;
       points[currentSize++] = r;
       points[currentSize++] = g;
       points[currentSize++] = b;

       theta+=(360/numPoints);

   }

   // For circle we will avoid printing out each vertex just because there
are too many
   printf("Drawing circle:\n");
   printf("RGB:(%lf %lf %lf)\n\n", r, g, b);

   // bind vao and vbo, buffer points
   bindVertexBufferAndArray();
   glBufferData(GL_ARRAY_BUFFER, sizeof(points), points, GL_STATIC_DRAW);

   // compile shaders
   compileShaders();

   // swap buffers, draw, then swap back
   glfwSwapBuffers(window);
   glDrawArrays(GL_TRIANGLE_FAN, 0, numPoints);
   glfwSwapBuffers(window);
```

```
    return;

}

/* Draws a rectangle with x and y as the bottom left point and h height, w
width
  We use element buffers and drawElement instead to simplify the number of
points
  needed (otherwise we would need to pass in 6 points for the 2 triangles).
*/
void drawRectangle(double x1, double y1, double x2, double y2, double x3,
double y3, double x4, double y4, char hex[]) {
    if (hex[0] == '#')
      hex++;
    int num = (int) strtol(hex, NULL, 16);
    double r = ((num >> 16) & 0xFF) / 255.0;
    double g = ((num >> 8) & 0xFF) / 255.0;
    double b = (num & 0xFF) / 255.0;

    double points[] = {
      x1, y1, 0.0, r, g, b,
      x2, y2, 0.0, r, g, b,
      x3, y3, 0.0, r, g, b,
      x4, y4, 0.0, r, g, b
    };

    printf("Drawing rectangle:\n");
    for (int i = 0; i < sizeof(points) / sizeof(double); i+= 6) {
      printf("Vertex: (%lf, %lf, %lf) RGB: (%lf, %lf, %lf)\n", points[i],
points[i+1],points[i+2],points[i+3],points[i+4],points[i+5]);
    }
    printf("\n");

    GLuint elements[] = {
        0, 1, 2,
        2, 3, 0
    };

    // bind vao and vbo, buffer points
    bindVertexBufferAndArray();
    glBufferData(GL_ARRAY_BUFFER, sizeof(points), points, GL_STATIC_DRAW);
```

```
    // bind ebo and buffer points
    bindElementBuffer();
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(elements), elements,
GL_STATIC_DRAW);

    // compile shaders
    compileShaders();

    glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
    return;
}


void drawLine(double startx, double starty, double endx, double endy, char
hex[]){
    if (hex[0] == '#')
      hex++;
    int num = (int) strtol(hex, NULL, 16);
    double r = ((num >> 16) & 0xFF) / 255.0;
    double g = ((num >> 8) & 0xFF) / 255.0;
    double b = (num & 0xFF) / 255.0;

    double points[] = {
        startx, starty, 0, r, g, b,
        endx, endy, 0, r, g, b
    };

    printf("Drawing line:\n");
    for (int i = 0; i < sizeof(points) / sizeof(double); i+= 6) {
      printf("Vertex: (%lf, %lf, %lf) RGB: (%lf, %lf, %lf)\n", points[i],
points[i+1],points[i+2],points[i+3],points[i+4],points[i+5]);
    }
    printf("\n");

    // bind vao and vbo, buffer points
    bindVertexBufferAndArray();
    glBufferData(GL_ARRAY_BUFFER, sizeof(points), points, GL_STATIC_DRAW);

    // set line width
    glLineWidth(10);
    glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
```

```
   // compile shaders
   compileShaders();

   // swap buffers, draw, then swap back
   glfwSwapBuffers(window);
   glDrawArrays(GL_LINES, 0, 2);
   glfwSwapBuffers(window);

   return;
}

/* Saves the current GLFW window to a given filepath */
void saveImage(char *filepath, GLFWwindow *w){
   int width, height;
   glfwGetFramebufferSize(w,&width,&height);
   GLsizei nrChannels = 3;
   GLsizei stride = nrChannels * width;
   stride += (stride % 4) ? (4-stride % 4) : 0;
   GLsizei bufferSize = stride * height;
   char buffer[bufferSize];
   glPixelStorei(GL_PACK_ALIGNMENT,4);
   glReadBuffer(GL_BACK);
   glReadPixels(0,0,width,height,GL_RGB,GL_UNSIGNED_BYTE,buffer);
   stbi_flip_vertically_on_write(1);
   stbi_write_png(filepath,width,height,nrChannels,buffer,stride);

   return;
}



/* Sets up the opening of a window:
   1. Creates the needed shaders
   2. Iterates through global vao array using index to bind and render all
the vertices
   3. Saves window as an image to img.png
*/
void jeomcRunAndSave() {

   /* Loop until the user closes the window */
   while (!glfwWindowShouldClose(window))
   {
```

```
        /* Poll for and process events */
        glfwPollEvents();
        glfwSwapBuffers(window);
    }

    // swap buffers right before saving image to preserve full image
    glfwSwapBuffers(window);

    saveImage("img.png", window);
    glfwTerminate();
    return;
}

#ifdef BUILD_TEST
int main(int argc, char *argv[]) {

 jeomcInit();

 drawCircle(0.0, -0.5, 0.4,"#FFFFFF");
 drawCircle(0.0, 0.1, 0.30,"#FFFFFF");
 drawCircle(0.0, 0.6, 0.20,"#FFFFFF");

 drawRectangle(-0.25,0.32,0.15,0.5,"B49EC9");

 drawCircle(-0.08,0.65,0.03, "#000000");
 drawCircle(0.08,0.65,0.03, "#000000");

 drawTriangle(0.0,0.6,0.15,0.58,0.0,0.56, "E0B949");

 drawCircle(0.0, 0.1, 0.03, "#000000");
 drawCircle(0.0, -0.1, 0.03, "#000000");
 drawCircle(0.0, -0.3, 0.03, "#000000");

 drawLine(-0.75,0.55,-0.18,0.3, "9A5415");
 drawLine(0.75,0.55,0.18,0.3, "9A5415");

 jeomcRunAndSave();
}
#endif
```

9.3 Build Scripts

9.3.1 testall.sh

```
#!/bin/sh

# Regression testing script for JEoMC
# 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

unameOut="$(uname -s)"
case "${unameOut}" in
   Linux*)     OS=Linux;;
   Darwin*)    OS=Mac;;
   CYGWIN*)    OS=Cygwin;;
   MINGW*)     OS=MinGw;;
   *)          OS="UNKNOWN:${unameOut}"
esac


# generate C linkers based on OS
if [ ${OS} = "Mac" ]; then
   # Mac OSX
   printf "\nYou are currently running on ${OS}, appropriate C flags will
be generated\n\n"
   FLAGS="-framework OpenGL -framework GLUT -lglfw -lGLEW"

   # Path to the LLVM interpreter
   LLI="/usr/local/opt/llvm@8/bin/lli"

   # Path to the LLVM compiler
   LLC="/usr/local/opt/llvm@8/bin/llc"
elif [ ${OS} = "Linux" ]; then
   # Assume Linux.
   printf "\nYou are currently running on ${OS}, appropriate C flags will
be generated\n\n"
   FLAGS="-lGL -lGLU -lglfw3 -lX11 -lXxf86vm -lXrandr -lpthread -lXi -ldl
-lXinerama -lXcursor -lm -lGLEW"

   # Path to the LLVM interpreter
   LLI="lli"
```

```
   # Path to the LLVM compiler
   LLC="llc"
else
   printf "\nYou are currently running on ${OS}, we do not support this
system\n\n"
fi

# Path to the C compiler
CC="cc"

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

# Set time limit for all operations
ulimit -t 30

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

keep=0

Usage() {
   echo "Usage: testall.sh [options] [.j files]"
   echo "-k    Keep intermediate files"
   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 -b $1 $2 ">" $3 1>&2
   diff -b "$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/.j//'`
   reffile=echo $1 | sed 's/.j$//'
   basedir="echo $1 | sed 's/\/[^\/]*$//'/."

   echo -n "$basename..."

   echo 1>&2
```

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

   generatedfiles=""

   generatedfiles="$generatedfiles ${basename}.ll ${basename}.s
${basename}.exe ${basename}.out" &&
   Run "$JEOMC" "$1" ">" "${basename}.ll" &&
   Run "$LLC" "-relocation-model=pic" "${basename}.ll" ">" "${basename}.s"
&&
   Run "$CC" "-o" "${basename}.exe" "${basename}.s" "jeomc_draw.o" "$FLAGS"
&&
   Run "./${basename}.exe" > "${basename}.out" &&
   Compare ${basename}.out ${reffile}.out ${basename}.diff

   # 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/.j//'`
   reffile=echo $1 | sed 's/.j$//'
   basedir="echo $1 | sed 's/\/[^\/]*$//'/."

   echo -n "$basename..."

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

   generatedfiles=""
```

```
   generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&
   RunFail "$JEOMC" "<" $1 "2>" "${basename}.err" ">>" $globallog &&
   Compare ${basename}.err ${reffile}.err ${basename}.diff

   # 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 kdpsh c; do
     case $c in
   k) # Keep intermediate files
       keep=1
       ;;
   h) # Help
       Usage
       ;;
     esac
done

shift expr $OPTIND - 1

LLIFail() {
 echo "Could not find the LLVM interpreter \"$LLI\"."
 echo "Check your LLVM installation and/or modify the LLI variable in
testall.sh"
 exit 1
}

which "$LLI" >> $globallog || LLIFail

if [ $# -ge 1 ]
then
```

```
   files=$@
else
   files="jeomc-tests/test-*.j jeomc-tests/fail-*.j microc-tests/fail-*.j
microc-tests/test-*.j"
fi

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

exit $globalerror
```

9.3.2 demo.sh

```
#!/bin/sh

# Regression testing script for JEoMC
# 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

unameOut="$(uname -s)"
case "${unameOut}" in
   Linux*)     OS=Linux;;
   Darwin*)    OS=Mac;;
   CYGWIN*)    OS=Cygwin;;
   MINGW*)     OS=MinGw;;
   *)          OS="UNKNOWN:${unameOut}"
esac
```

```
# generate C linkers based on OS
if [ ${OS} = "Mac" ]; then
   # Mac OSX
   printf "\nYou are currently running on ${OS}, appropriate C flags will
be generated\n\n"
   FLAGS="-framework OpenGL -framework GLUT -lglfw -lGLEW"

   # Path to the LLVM interpreter
   LLI="/usr/local/opt/llvm@8/bin/lli"

   # Path to the LLVM compiler
   LLC="/usr/local/opt/llvm@8/bin/llc"
elif [ ${OS} = "Linux" ]; then
   # Assume Linux.
   printf "\nYou are currently running on ${OS}, appropriate C flags will
be generated\n\n"
   FLAGS="-lGL -lGLU -lglfw3 -lX11 -lXxf86vm -lXrandr -lpthread -lXi -ldl
-lXinerama -lXcursor -lm -lGLEW"

   # Path to the LLVM interpreter
   LLI="lli"

   # Path to the LLVM compiler
   LLC="llc"
else
   printf "\nYou are currently running on ${OS}, we do not support this
system\n\n"
fi

# Path to the C compiler
CC="cc"

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

# Set time limit for all operations
ulimit -t 30

globallog=testall.log
```

```
rm -f $globallog
error=0
globalerror=0

keep=0

Usage() {
   echo "Usage: testall.sh [options] [.j files]"
   echo "-k    Keep intermediate files"
   echo "-h    Print this help"
   exit 1
}

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

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

Check() {
   error=0
   basename=`echo $1 | sed 's/.*\\///
                            s/.j//'`
   reffile=echo $1 | sed 's/.j$//'
   basedir="echo $1 | sed 's/\/[^\/]*$//'/."

   echo -n "$basename..."

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

```
   generatedfiles=""

   generatedfiles="$generatedfiles ${basename}.ll ${basename}.s
${basename}.exe ${basename}.out" &&
   Run "$JEOMC" "$1" ">" "${basename}.ll" &&
   Run "$LLC" "-relocation-model=pic" "${basename}.ll" ">" "${basename}.s"
&&
   Run "$CC" "-o" "${basename}.exe" "${basename}.s" "jeomc_draw.o" "$FLAGS"
&&
   Run "./${basename}.exe" > "${basename}.out" &&

   # 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 kdpsh c; do
     case $c in
   k) # Keep intermediate files
       keep=1
       ;;
   h) # Help
       Usage
       ;;
     esac
done

shift expr $OPTIND - 1

LLIFail() {
 echo "Could not find the LLVM interpreter \"$LLI\"."
 echo "Check your LLVM installation and/or modify the LLI variable in
```

```
testall.sh"
 exit 1
}

which "$LLI" >> $globallog || LLIFail

if [ $# -ge 1 ]
then
   files=$@
else
   files="demo/*.j"
fi

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

exit $globalerror
```

9.3.2 Makefile

```
# "make test" Compiles everything and runs the regression tests
.PHONY : test
test : all testall.sh
   ./testall.sh

# "make test" Compiles everything and runs the drawing demos
.PHONY : demo
demo : all demo.sh
./demo.sh
```

```
# "make all" builds the executable as well as the "draw" library designed
# to test linking external code
.PHONY : all
all : jeomc.native jeomc_draw.o

# "ocamlbuild calc.native" will also build the calculator
#before: opam config exec --
jeomc.native :
   opam exec -- \
   ocamlbuild -use-ocamlfind jeomc.native

# "make clean" removes all generated files
.PHONY : clean
clean :
   ocamlbuild -clean
   rm -rf testall.log ocamlllvm *.diff *.err *.ll *.s *.exe *.out
   rm -f *.o img.png jeomc_draw
```

9.4 Tests

Tests from jeomc-tests folder

9.4.1 fail-break.j

```
int main(){
    int j;
    int i;
    j = 0;

    for (i=0;i<5;i=i+1){
      if (i == 3) {
        break;
        j = 1;
      }
        j = j+i;
    }
    print(j);
    return 0;
}
```

9.4.2 fail-continue.j

```
int main(){
    int j;
    int i;
    j = 0;

    for (i = 0; i < 5; i = i + 1){
    if (i == 3) {
      i = i+1;
      continue;
      i = 2;
    }
        j = j+i;
    }
    print(j);
    return 0;
}
```

9.4.3 fail-draw-func1.j

```
/*
Fail: Trying to supply an argument to a function that expects none
*/

int main()
{
  float x;
  float y;
  float f;

  x = 0.5;
  y = 0.5;
  f = 0.5;

  jeomcInit(x);

  return 0;
}
```

9.4.4 fail-draw-func2.j

```
/*
Fail: Trying to supply too few arguments when func expects 7
*/

int main()
{
  float x;
  float y;
  float f;

  x = 0.5;
  y = 0.5;
  f = 0.5;

  jeomcInit();

  drawTriangle(x, y);

  jeomcRunAndSave();

  return 0;
}
```

9.4.5 fail-draw-func3.j

```
/*
Fail: Trying to supply invalid arguments (float when the last arg needs to
be string hexcode)
*/

int main()
{
  float x;
  float y;
  int f;

  x = 0.5;
  y = 0.5;
  f = 1;

  jeomcInit();
```

```
  drawTriangle(x, y, x, y, x, y, x);

  jeomcRunAndSave();

  return 0;
}
```

9.4.6 fail-string.j

```
int main(){
    string j;

    j = "a"a;
    prints(j);

    return 0;
}
```

9.4.7 fail-string-int-comparison.j

```
int main(){
    string j;
    int i;

    i = 1;
    j = "1";
    if (j == i) {
        print(1);
    }

    return 0;
}
```

9.4.8 test-arith4.j

```
int main()
{
  print(1 + 4 / 2 + 3);
  return 0;
}
```

9.4.9 test-arith5.j

```
int foo(int a)
{
  return a;
}

int main()
{
  int a;
  a = 42;
  a = a / 6;
  print(a);
  return 0;
}
```

9.4.10 test-bool.j

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

      a = true;
      b = false;


      if(a){
            prints("A is true");
      }
      if(b){
            prints("B is true");
      }


      return 0;
}
```

9.4.11 test-bool2.j

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

```
	a = true;
	b = false;

	if(a==b){
		prints("matched");
	}

	if(a!=b){
		prints("mismatched");
	}

	return 0;

}
```

9.4.12 test-bool3.j

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

	a = true;
	b = 2;

	if(a == true && b == 2){
		print(2);
	}

	else{
		print(4);
	}

	return 0;
}
```

9.4.13 test-break.j

```
int main(){
    int j;
    int i;
```

```
    j = 0;
    for (i=0;i<5;i=i+1){
      if (i == 3) {
        break;
      }
        j = j+i;
    }
    print(j);
    return 0;
}
```

9.4.14 test-break2.j

```
int main(){
    int i;
    for (i=0;i<5;i=i+1){
      if (i == 3) {
        break;
      }
      print(i);
    }
    return 0;
}
```

9.4.15 test-continue.j

```
int main(){
    int j;
    int i;
    j = 0;
    for (i=0;i<5;i=i+1){
    if (i == 3) {
      i = i+1;
      continue;
    }
        j = j+i;
    }
    print(j);
    return 0;
}
```

9.4.16 test-continue2.j

```
int main(){
    bool b;
    b = true;
    while(b) {
        print(1);
        b = false;
        if (true) {
            continue;
        }
        print(2);
    }
    print(3);
    return 0;
}
```

9.4.17 test-draw1.j

```
/* test hex code translation is working properly
   do not run&save the window at the end
*/
int main()
{

  jeomcInit();

  drawRectangle(-0.25, 0.47 , 0.25,0.47,   0.25,0.32,  -0.25,0.32
,"B49EC9");

  return 0;
}
```

9.4.18 test-draw2.j

```
/* test hex code translation is working properly
   do not run&save the window at the end
*/
int main()
{
```

```
  jeomcInit();

  drawTriangle(0.0,0.6,0.15,0.58,0.0,0.56, "E0B949");

  return 0;
}
```

9.4.19 test-empty-string.j

```
int main(){
    string j;
    string k;

    j = "";
    k = "";

    if (j == "") {
      print(1);
      prints(j);
    }

    return 0;
}
```

9.4.20 test-for-add.j

```
int main(){
    int j;
    int i;
    j = 0;
    for (i=0;i<5;i=i+1){
        j = j+i;
    }
    print(j);
    return 0;
}
```

9.4.21 test-for-and.j

```
int main(){
```

```
  string vowels;
  int numbers;

  vowels = "aeiou";
  numbers = 26;

/* Testing && with ints and strings */

  if(vowels == "aeiou" && numbers == 26){
    prints("There are 26 letters in the alphabet, including the vowels
aeiou");
  }

  return 0;

}
```

9.4.21 test-for-div.j

```
int main(){
    int j;
    int i;
    j = 48;
    for (i=1;i<5;i=i+1){
        j = j/i;
    }
    print(j);
    return 0;
}
```

9.4.22 test-for-minus.j

```
int main(){
    int j;
    int i;
    j = 30;
    for (i=0;i<5;i=i+1){
        j = j-i;
    }
    print(j);
    return 0;
}
```

9.4.23 test-for-mul.j

```
int main(){
    int j;
    int i;
    j = 2;
    for (i=1;i<4;i=i+1){
        j = j*i;
    }
    print(j);
    return 0;
}
```

9.4.24 test-hello2.j

```
int main()
{
  print(21);
  prints("hello");
  return 0;
}
```

9.4.25 test-hex1.j

```
/* test hex code translation is working properly
   do not run&save the window at the end
*/
int main()
{

  jeomcInit();

  drawCircle(0.0, -0.5, 0.4,"#FFFFFF");
  drawCircle(0.0, -0.5, 0.4,"#9E2828");

  return 0;
}
```

9.4.26 test-hex2.j

```
/* test hex code translation with various ways of passing in strings
   You can lead with # or omit it. We take care of both cases
 */

int main()
{

  jeomcInit();

  /* working examples */
  drawCircle(0.0, -0.5, 0.4,"#FFFFFF");
  drawCircle(0.0, -0.5, 0.4,"FFFFFF");
  drawCircle(0.0, -0.5, 0.4,"#FFFFFF       ");

  /*
  Note that leading whitespace and whitespace in between strings will NOT work
  drawCircle(0.0, -0.5, 0.4,"      #FFFFFF");
  drawCircle(0.0, -0.5, 0.4,"#FFFF      FF");
  */

  return 0;
}
```

9.4.27 test-hex3.j

```
/* test hex code translation when we pass in "bad" strings */

int main()
{

  jeomcInit();

  /* should only read the first 6 chars, not including # */
  drawCircle(0.0, -0.5, 0.4,"#FFFFFFFFFFFFFFF");

  /* should be 0,0,0 white */
  drawCircle(0.0, -0.5, 0.4,"");

  /* should be 0,0,0 white */
  drawCircle(0.0, -0.5, 0.4,"            ");
```

```
  /* should only read the first 6 chars */
  drawCircle(0.0, -0.5, 0.4,"FFFFFF        HELLO WORLD");

  /* should be empty white */
  drawCircle(0.0, -0.5, 0.4,"HELLO WORLD");

  /* should be read same as #FFFF */
  drawCircle(0.0, -0.5, 0.4,"#FFFFHELLO WORLD");

  return 0;
}
```

9.4.28 test-hex4.j

```
/* test hex code translation when we pass in "bad" strings */

int main()
{

  jeomcInit();

  /* Once we encounter an invalid hex character, we stop reading even if
valid characters come after */
  /* these two should produce identical RGBs */
  drawCircle(0.0, -0.5, 0.4,"#Fz0000");
  drawCircle(0.0, -0.5, 0.4,"#F");

  return 0;
}
```

9.4.29 test-string.j

```
int main(){
    string j;
    string k;
    string l;
    j = "abc";
    k = "abc";
    l = "ab";

    if (j == k) {
      print(1);
```

```
    }

    if (j == l) {
      print(2);
    }

    if (k == l) {
      print(3);
    }

    return 0;
}
```

9.4.30 test-string-case.j

```
int main(){
    string j;
    string k;
    string l;
    string m;

    j = "FOO";
    k = "foo";
    l = "FoO";
    m = "foo";

    if (j == k) {
      print(1);
    }

    if (j == l) {
      print(2);
    }

    if (j == m) {
      print(3);
    }

    if (k == l) {
      print(4);
    }
```

```
    if (k == m) {
      print(5);
    }

    return 0;
}
```

9.4.31 test-string2.j

```
int main(){
    string j;

    j = "Hello World!";
    prints(j);

    return 0;
}
```

9.4.32 test-string3.j

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

    a = "Abc";
    b = "aBc";



    if (a == b) {
      prints("matched");
    }

    if (a != b) {
      prints("mismatched");
    }

    return 0;
}
```

9.5 Git Log

```
commit 933e6d6b24fd63ac5de8c8914be093df569642e0 (HEAD -> master,
origin/master, origin/HEAD)
Author: lujeremy <jeremylu04@gmail.com>
Date:   Wed Dec 22 00:08:31 2021 -0500

    Add a demo directory and script

commit 31d8fe3c83863d2ec7bf341e4e88e6639b05c1fd
Merge: d60b19b 0a14ff5
Author: ebschwartz3 <es3962@barnard.edu>
Date:   Tue Dec 21 22:40:11 2021 -0500

    Merge branch 'master' of https://github.com/lujeremy/JEoMC

commit d60b19bdb604d3e34883ce733f9dd940c393e08d
Author: ebschwartz3 <es3962@barnard.edu>
Date:   Tue Dec 21 22:40:08 2021 -0500

    New Tests

    New tests
    - booleans
    - strings
    - ints

commit 0a14ff5d4413454e8fa569342f66f934b7ad7003 (HEAD -> master,
origin/master, origin/HEAD)
Merge: d6e3436 683b96b
Author: conniez <connieeez.97@gmail.com>
Date:   Tue Dec 21 22:23:17 2021 -0500

    Merge branch 'master' of https://github.com/lujeremy/JEoMC
    Diff tests

commit d6e34367cf0da75f4c910e61e2e7eec049731db3
Author: conniez <connieeez.97@gmail.com>
Date:   Tue Dec 21 22:22:58 2021 -0500

    all arithmetic
```

```
commit 683b96b32559e3a12b14f394aa37fea36466480b
Author: lujeremy <jeremylu04@gmail.com>
Date:   Tue Dec 21 22:05:00 2021 -0500

    Move tests from microc -> jeomc

commit dbebd0d634cdad73338c2052f520e4144b513a08
commit 0a14ff5d4413454e8fa569342f66f934b7ad7003 (HEAD -> master,
origin/master, origin/HEAD)
Merge: d6e3436 683b96b
Author: conniez <connieeez.97@gmail.com>
Date:   Tue Dec 21 22:23:17 2021 -0500

    Merge branch 'master' of https://github.com/lujeremy/JEoMC
    Diff tests

commit d6e34367cf0da75f4c910e61e2e7eec049731db3
Author: conniez <connieeez.97@gmail.com>
Date:   Tue Dec 21 22:22:58 2021 -0500

    all arithmetic

commit 683b96b32559e3a12b14f394aa37fea36466480b
Author: lujeremy <jeremylu04@gmail.com>
Date:   Tue Dec 21 22:05:00 2021 -0500

    Move tests from microc -> jeomc

commit dbebd0d634cdad73338c2052f520e4144b513a08
Author: lujeremy <jeremylu04@gmail.com>
Date:   Tue Dec 21 21:58:33 2021 -0500

    Remove unnecessary files

commit 866092409a25bada37031b9a187e248f8c018166
Merge: 74b4865 d2a5a06
Author: lujeremy <jeremylu04@gmail.com>
Date:   Tue Dec 21 21:56:20 2021 -0500

    Merge branch 'master' of https://github.com/lujeremy/JEoMC

commit 74b4865fef21e35cdcdc540786ac9a5246e87b30
```

```
Author: lujeremy <jeremylu04@gmail.com>
Date:   Tue Dec 21 21:56:06 2021 -0500

    Draw rectangles with points

commit d2a5a06a4feb4fec6fb7de81ae5aaab207890239
Author: conniez <connieeez.97@gmail.com>
Date:   Tue Dec 21 21:51:05 2021 -0500

    more tests

commit 030e3dd0d5b9172ba8b6409dc03b26b401f1357e
Author: lujeremy <jeremylu04@gmail.com>
Date:   Tue Dec 21 21:22:09 2021 -0500

    Fix outdated tests

commit fac4d69756be2088a3ce036ecb37146d4b0a15f2
Author: lujeremy <jeremylu04@gmail.com>
Date:   Tue Dec 21 21:19:02 2021 -0500

    Edit swapbuffer locations

commit bd63d993ccec4da86763066fbb6c4174c7dbb77e
Author: lujeremy <jeremylu04@gmail.com>
Date:   Tue Dec 21 20:26:57 2021 -0500

    Add tests for draw and string comparison

commit 86ca47b88ff07ed33be7e572666641ef545aa83a
Author: lujeremy <jeremylu04@gmail.com>
Date:   Tue Dec 21 20:03:39 2021 -0500

    Add hex-conversion related tests

commit 5071eeefa73c22e7bf0f5eb7d9abe23aa63891cb
Author: lujeremy <jeremylu04@gmail.com>
Date:   Tue Dec 21 19:06:01 2021 -0500

    Add debugging prints to drawing functions

commit a33f1827c562fe373598680e41769fbd6767ecc2
```

```
Author: ebschwartz3 <es3962@barnard.edu>
Date:   Tue Dec 21 18:05:58 2021 -0500

    err

commit 03186ed9b5a58ef112adcd64c6dbbade7860dfa6
Author: lujeremy <jeremylu04@gmail.com>
Date:   Tue Dec 21 18:01:40 2021 -0500

    Add test output to gitignore

commit e0970edc96da3cc36309906e8a9cb117c2989f1f
Author: lujeremy <jeremylu04@gmail.com>
Date:   Mon Dec 20 23:20:05 2021 -0500

    Add support for hexcode colors

commit 079036d00e8016440405b061620afa154e2ee730
Merge: 076513d 6488261
Author: lujeremy <jeremylu04@gmail.com>
Date:   Mon Dec 20 23:13:30 2021 -0500

    Merge branch 'master' of https://github.com/lujeremy/JEoMC

commit 076513d4ffcc56602b6064506feb80217e7db3a3
Author: lujeremy <jeremylu04@gmail.com>
Date:   Mon Dec 20 23:12:04 2021 -0500

    Add support for hexcode colors

commit 6488261ab1056e988d9a952acaf362173bf983f7
Merge: f39da50 219451f
Author: Jeremy Lu <jeremylu04@gmail.com>
Date:   Mon Dec 20 22:10:44 2021 -0500

    Merge pull request #7 from lujeremy/shader-cleanup

    Shader cleanup

commit 219451fab5e5bdf70de973f207d03ed144460557
Author: lujeremy <jeremylu04@gmail.com>
Date:   Mon Dec 20 22:10:07 2021 -0500
```

```
    Fix image saving swapBuffer; Finish adding drawLine to ocaml; Fix tests

commit 89111906b4682adb1f06c576fd2febeee98085c7
Author: lujeremy <jeremylu04@gmail.com>
Date:   Mon Dec 20 21:20:57 2021 -0500

    Set colors via array

commit f39da50b8b9aa273c651a82f0125981f446f09d7
Merge: 1f281d7 c56c9bc
Author: Jeremy Lu <jeremylu04@gmail.com>
Date:   Mon Dec 20 20:36:44 2021 -0500

    Merge pull request #6 from lujeremy/melodyhsu47-patch-1

    Melodyhsu47 patch 1

commit c56c9bc49bcbd62417506276398bb50ebdfaa064
(origin/melodyhsu47-patch-1)
Author: lujeremy <jeremylu04@gmail.com>
Date:   Mon Dec 20 20:36:08 2021 -0500

    Update shaders

commit 06435a3c84543908adccc220de2d64533fc24740
Author: lujeremy <jeremylu04@gmail.com>
Date:   Mon Dec 20 20:10:17 2021 -0500

    Vertex colors

commit ade7f0679ccd3122f9d71a0f8010816aacf2d722
Author: lujeremy <jeremylu04@gmail.com>
Date:   Mon Dec 20 19:41:22 2021 -0500

    More merge

commit 303a424fb36b0d3cb8cd0836faf0b7bf8fc0b93a
Merge: 0ab747a 1f281d7
Author: lujeremy <jeremylu04@gmail.com>
Date:   Mon Dec 20 19:32:12 2021 -0500
```

```
    Merge branch 'master' into melodyhsu47-patch-1

commit 0ab747a27051f0f3b9b1802f72b78f6a6071228d
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Mon Dec 20 17:38:44 2021 -0500

    Update test-draw-save.j

commit 1f281d7b76d39d8eba1c051ab1630cc6e5ceac73
Author: lujeremy <jeremylu04@gmail.com>
Date:   Mon Dec 20 15:11:37 2021 -0500

    Add more string tests

commit 9ebae7413401d29f22d934357f6ad84f252cef08
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Mon Dec 20 14:50:23 2021 -0500

    draw utils

commit 834ef40badec9d95dfa7d3f2d8e477637512b0af
Merge: de61733 971a864
Author: ebschwartz3 <es3962@barnard.edu>
Date:   Mon Dec 20 14:47:51 2021 -0500

    Merge branch 'master' of https://github.com/lujeremy/JEoMC

commit de61733622327d2b11f125d6cdec379f6e1ac0ac
Author: ebschwartz3 <es3962@barnard.edu>
Date:   Mon Dec 20 14:47:47 2021 -0500

    PrintS

    Added prints (print string) to codegen and semant

commit 971a864c95e84642fe80baf473426fe3289d75c8
Author: lujeremy <jeremylu04@gmail.com>
Date:   Mon Dec 20 14:36:44 2021 -0500

    Add a string test

commit 4cf176234334fcf2e26732989533e9f863575469
```

```
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Mon Dec 20 14:35:51 2021 -0500

    adding additional default functions

commit 486cbe8952a4905f3d903d427fb8a8e263bc31fb
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Mon Dec 20 14:34:32 2021 -0500

    added default functions

commit c2dcb9a0ce1bd0b7f57538e103a0286e34bafa95
Author: lujeremy <jeremylu04@gmail.com>
Date:   Mon Dec 20 11:29:37 2021 -0500

    Add back removed changes

commit 6af6c39d59b61f8f508324fd00416d56f7e365c3
Author: ebschwartz3 <71985184+ebschwartz3@users.noreply.github.com>
Date:   Mon Dec 20 00:59:35 2021 -0500

    Update parser.mly

commit 4b09a3614610ef2fee4307d3792b3c8cd6c79e41
Author: ebschwartz3 <71985184+ebschwartz3@users.noreply.github.com>
Date:   Mon Dec 20 00:58:45 2021 -0500

    Update semant.ml

commit dad5bec29673948b6e70cd171e43b299e8b2b63c
Author: ebschwartz3 <71985184+ebschwartz3@users.noreply.github.com>
Date:   Mon Dec 20 00:57:46 2021 -0500

    Update scanner.mll

commit 5e395a19aca7d8cd6e2114bc12aeea95add713df
Author: ebschwartz3 <71985184+ebschwartz3@users.noreply.github.com>
Date:   Mon Dec 20 00:57:10 2021 -0500

    Update sast.ml

commit eabd99b0a81eb1078ce72cee3e7b841d2b091b78
```

```
Author: ebschwartz3 <71985184+ebschwartz3@users.noreply.github.com>
Date:   Mon Dec 20 00:56:30 2021 -0500

    Update codegen.ml

    String added

commit 7bdd38d0931214ed08b4f22d70f999336b394630
Author: ebschwartz3 <71985184+ebschwartz3@users.noreply.github.com>
Date:   Mon Dec 20 00:55:39 2021 -0500

    Update ast.ml String

commit fc817f0882c61d6eee641ffc84751968f24faab2
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Sun Dec 19 20:09:12 2021 -0500

    fix typo in apple ifdef

commit 874874a24c6dc36887f5b01feb86c604da6e59fc
Merge: bda6881 e6de17d
Author: Jeremy Lu <jeremylu04@gmail.com>
Date:   Sun Dec 19 18:10:15 2021 -0500

    Merge pull request #5 from lujeremy/feature/colors-shapes

    Add function to allow drawing with different colors and shape

commit e6de17dd680aff3081555127ccb2e88d07ab3ccd
(origin/feature/colors-shapes)
Author: lujeremy <jeremylu04@gmail.com>
Date:   Sun Dec 19 18:09:21 2021 -0500

    Separate jeomc tests from microc tests

commit 8d7409d47bace2d6146f7df4a14cb35f7504753b
Author: lujeremy <jeremylu04@gmail.com>
Date:   Sun Dec 19 18:00:57 2021 -0500

    Update tests

commit ba31f505a5d91efba3e139c72c52acbb2a6854e4
```

```
Author: lujeremy <jeremylu04@gmail.com>
Date:   Sun Dec 19 17:55:09 2021 -0500

    Remove unused files

commit d9204fad6aaf921e093fa0f37514bf18d1337d56
Author: lujeremy <jeremylu04@gmail.com>
Date:   Sun Dec 19 17:54:16 2021 -0500

    Add new function tests; port over microC tests to different directory

commit 9e40480a6380f901241f0690bd01a1c0b493ec1b
Author: lujeremy <jeremylu04@gmail.com>
Date:   Sun Dec 19 16:07:18 2021 -0500

    Remove extra file

commit 447161db23955fa55f6edc188935a00577b20fce
Author: lujeremy <jeremylu04@gmail.com>
Date:   Sun Dec 19 16:06:36 2021 -0500

    Integrate rectangle, circles, and colors to OCaml

commit 04a7a0e3bc576498c5ce1656b17e0103b240a8f4
Author: lujeremy <jeremylu04@gmail.com>
Date:   Sun Dec 19 15:25:47 2021 -0500

    Add ability to draw circles in C

commit f8a0bc5b15daea201757b2bee9a1d4116141b6d4
Author: lujeremy <jeremylu04@gmail.com>
Date:   Sun Dec 19 14:46:15 2021 -0500

    Add colors and rectangle drawing to C program

commit bda6881832900087ff6090d574b130f26a3c72bb
Merge: 0e646b0 5266e17
Author: Jeremy Lu <jeremylu04@gmail.com>
Date:   Sun Dec 19 11:07:06 2021 -0500

    Merge pull request #4 from lujeremy/point-generation
```

```
    Add ability to draw triangles with functions

commit 5266e171ffb1d1a2beb46f9737a3d268fcebea13
Author: lujeremy <jeremylu04@gmail.com>
Date:   Sun Dec 19 11:05:59 2021 -0500

    Change vertex pointer to GL_DOUBLE

commit b12f850f05e87dea9b3e35bfa4fd1b1efadc1c55
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Sun Dec 19 00:09:02 2021 -0500

    add multiple triangle draw

commit 86b6c865e196f9f9eb04e4b91893643f42f54468
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Sat Dec 18 16:30:11 2021 -0500

    float points[] -> double points[]

commit 8f276a0af57fb2075eb4731f013b686163ec3793
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Sat Dec 18 16:27:07 2021 -0500

    Update drawTriangle.c

commit 9d2dd3ea4382b5c84a719dc8a9b09cb33be2937c
Author: lujeremy <jeremylu04@gmail.com>
Date:   Sat Dec 18 11:01:04 2021 -0500

    Add doc comments to drawTriangle.c

commit d7e46afe6fcbdc857166531e28559baa9f036675
Author: lujeremy <jeremylu04@gmail.com>
Date:   Sat Dec 18 10:45:37 2021 -0500

    Edit float to be double

commit 101061fc69620a8d7731ce70723335ce0bcb84e6
Author: lujeremy <jeremylu04@gmail.com>
Date:   Fri Dec 17 19:27:54 2021 -0500
```

```
    Remove unnecessary draw.c and printbig

commit c5b92faa8608d45ba93e36a80461270c88689cf7
Author: lujeremy <jeremylu04@gmail.com>
Date:   Fri Dec 17 19:19:31 2021 -0500

    Integrate drawing functions to compiler in OCaml

commit d123f94ef3e8f28e1bf09e5ea3452c14e1d81d21
Author: lujeremy <jeremylu04@gmail.com>
Date:   Fri Dec 17 18:46:38 2021 -0500

    Abstract drawing functions so that they can each be called individually

commit d262644758832ff3bcb107cc57b7d4a21be77de4
Author: lujeremy <jeremylu04@gmail.com>
Date:   Fri Dec 17 18:08:44 2021 -0500

    Split drawing program into multiple functions

commit b8670a6aaf612475b9776996984a23487660f246 (origin/point-generation)
Author: lujeremy <jeremylu04@gmail.com>
Date:   Thu Dec 16 17:52:41 2021 -0500

    Add sample C code for drawing multiple triangles

commit 0e646b0f3cdb89951eb4995f6b4b1c2839117c4c (array)
Merge: ec2a335 b1a6e34
Author: Jeremy Lu <jeremylu04@gmail.com>
Date:   Wed Dec 15 16:01:07 2021 -0500

    Merge pull request #3 from lujeremy/feature/continue-and-break

    Add continue and break keywords

commit b1a6e3418269e2c0bb2dbc3b1085e46eabbcd75e
Author: lujeremy <jeremylu04@gmail.com>
Date:   Wed Dec 15 15:43:53 2021 -0500

    Add tests for break and continue

commit 42f3231ed2abad9226e2ea6036054494d7e322aa
```

```
Author: lujeremy <jeremylu04@gmail.com>
Date:   Wed Dec 15 15:21:57 2021 -0500

    Implement Continue and Break keywords

commit ec2a335aeebc71d46c62a931f1f7afd5f2240415
Author: lujeremy <jeremylu04@gmail.com>
Date:   Tue Dec 14 22:42:30 2021 -0500

    Update lli and llc to be based on os

commit faf602eb5a8ceafd492c7cffadf8b44d32a07d3f
Merge: ab320c0 211cc0d
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Mon Dec 13 22:19:24 2021 -0500

    Merge pull request #2 from lujeremy/melodyhsu47-patch-1

    Add drawing/saving functionality

commit 211cc0d84d1e759eea812a03f2431c2a52779822
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Mon Dec 13 22:10:50 2021 -0500

    Update testall.sh

commit 2b78e7b71b7ac0b9b232bc8f14fa496e3c34fcdc
Author: lujeremy <jeremylu04@gmail.com>
Date:   Mon Dec 13 15:48:09 2021 -0500

    Change bash script and C files to support both mac & linux

commit ac0fce683a57bc21efbd5486b42d55c9373a2078
Merge: b009883 ab320c0
Author: lujeremy <jeremylu04@gmail.com>
Date:   Mon Dec 13 14:24:13 2021 -0500

    Merge branch 'master' into melodyhsu47-patch-1

commit ab320c054c687b53ab2133f7a4ff1d2d42a5bf5a
Merge: cbe45b4 76db389
Author: Jeremy Lu <jeremylu04@gmail.com>
```

```
Date:   Mon Dec 13 13:57:49 2021 -0500

    Merge pull request #1 from lujeremy/microc-port

    Microc port

commit b009883f7401caeb19a7fe32a045ee6c9ddb132f
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Mon Dec 13 10:56:29 2021 -0500

    Add files via upload

commit b25a4c10e6818c7d9cb984e60bbdd25a7c4babda
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Mon Dec 13 10:55:37 2021 -0500

    Create test-draw-save.out

commit 446f582a9f31f079031c849d760b205989da6935
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Mon Dec 13 10:55:16 2021 -0500

    Create test-draw.out

commit de33167a705acf658db205ecb83c124335ac2369
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Mon Dec 13 10:54:49 2021 -0500

    Add files via upload

commit f1dacc180524c4a1f67e789440a27cde5bf9453e
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Mon Dec 13 10:53:52 2021 -0500

    Add files via upload

commit a505cd631f7d18a959b17150dd8d50a83345bf91
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Mon Dec 13 10:53:21 2021 -0500

    Update and rename test1 to fail-assign1.err
```

```
commit f8ca516472fa29e1733198242743cbfe1ee31b2d
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Mon Dec 13 10:52:17 2021 -0500

    Create test1

commit 2d218149f89a4ab33484f8bdac084272453e4ee4
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Mon Dec 13 10:42:05 2021 -0500

    Add files via upload

commit d73b117b40f08d397fa4edaa78bce065f872b7aa
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Mon Dec 13 10:41:13 2021 -0500

    Add files via upload

commit a0304caa9986108bb6d7f402b1dff74d6ffdd388
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Mon Dec 13 10:40:49 2021 -0500

    Update scanner.mll

commit 6142876eca28a56296e2a111a177f5e954f47ec1
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Mon Dec 13 10:40:07 2021 -0500

    Update parser.mly

commit 9b23cde45c0139f3dca83870da9e13d31d6cea60
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Mon Dec 13 10:39:46 2021 -0500

    Delete jeomcparse.mly

commit 66f66ee139b2230d4b58ed5b37763db9fbcfba86
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Mon Dec 13 10:38:19 2021 -0500

    Add files via upload
```

```
commit f40d3186c71b8547b7f79371b04453f3905edacc
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Mon Dec 13 10:36:56 2021 -0500

    Update and rename calc.ml to jeomc.ml

commit 5cee5c6e64789d7824e5ed7fee1e48ccb05914bd
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Mon Dec 13 10:36:19 2021 -0500

    Delete ast.mli

commit 757636f833a1f9d0ac6fc403e1f51c8b5b3e42dd
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Mon Dec 13 10:35:39 2021 -0500

    Update ast.ml

commit 76db38999c244dbbfbb16d0c0ad772cdc8ef8f06
Author: lujeremy <jeremylu04@gmail.com>
Date:   Wed Dec 1 16:27:13 2021 -0500

    Remove unused files

commit 7c4f802300060f0aa3b7a831647e8b9e3d91f73d
Author: lujeremy <jeremylu04@gmail.com>
Date:   Wed Dec 1 16:25:34 2021 -0500

    Remove submodule glfw

commit 126021ea37231087980556fd96facf23ac495191
Author: Jeremy Lu <jeremylu04@gmail.com>
Date:   Wed Dec 1 16:16:19 2021 -0500

    Update README.md

commit 7714870476fc046f20cd3e6cfc919eef6a857faf
Author: lujeremy <jeremylu04@gmail.com>
Date:   Wed Dec 1 15:45:11 2021 -0500

    Revise dockerfile, gitignore
```

```
commit aa0dba3fe233afa361b8489ee831bd58f0c5f73b
Author: lujeremy <jeremylu04@gmail.com>
Date:   Sat Nov 27 18:57:31 2021 -0500

    Link draw to our code

commit 0fcb08f1bf68c9a4726e80c8cd565931a917ac9d
Author: lujeremy <jeremylu04@gmail.com>
Date:   Sat Nov 27 18:21:19 2021 -0500

    Name changes to JEoMC

commit ab7b3013f76c9b85e9dc432226f34890445580a5
Author: lujeremy <jeremylu04@gmail.com>
Date:   Sat Nov 27 18:12:56 2021 -0500

    Port working microC

commit cbe45b43ecb6d3d2e14034af8713811db0310e78
Author: lujeremy <jeremylu04@gmail.com>
Date:   Sat Nov 27 17:39:09 2021 -0500

    Add testing

commit 1cf95269d8f63fcfb2ee5847999a1d11ed5e58a7
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Sat Nov 27 12:07:59 2021 -0500

    Update and rename ast.mli to ast.ml

commit 5287d9063b493fdb4f2452240a79cbed4f38e3fe
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Sat Nov 27 12:07:32 2021 -0500

    Update sast.ml

commit bd5f8ee8558a1e0e9f44e8c056628ea686d1cfff
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Sat Nov 27 12:07:07 2021 -0500

    Add files via upload
```

```
commit 302f56644739df42d0a9c21933a3a5f1cc7525d9
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Sat Nov 27 12:05:37 2021 -0500

    Update parser.mly

commit 13b7eb51b6be658cb88a61309962e3c618041f1a
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Sat Nov 27 12:05:07 2021 -0500

    Update jeomc.ml

commit 4083769bef82c43b5909ff844e0fd5bcab3861b4
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Sat Nov 27 12:04:40 2021 -0500

    Update semant.ml

commit c7b1dcc3a8fab0ed4e52e46a1bfae2bdda1ee328
Author: Jeremy Lu <jeremylu04@gmail.com>
Date:   Thu Nov 25 15:54:30 2021 -0500

    Update README.md

commit 0725eabbe875995f051341b101cf6eec12c9c0a9
Author: lujeremy <jeremylu04@gmail.com>
Date:   Thu Nov 25 15:49:59 2021 -0500

    Add glfw as submodule

commit 91b1451431fdf021c39960fb60b22705606772b7
Author: lujeremy <jeremylu04@gmail.com>
Date:   Thu Nov 25 15:46:44 2021 -0500

    Update gitignore, Add cmake

commit 6273a4c115d6803bb91e9347e641080179e8d87c
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Sun Nov 21 14:22:20 2021 -0500

    Update scanner.mll
```

```
commit c4d18f002ad39875b51e9c3f0344bf371b2cc3f8
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Sun Nov 21 14:21:26 2021 -0500

    Update ast.mli

commit 34c127479b34afb60a19a6a30504f5c1d6871f27
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Sun Nov 21 14:00:45 2021 -0500

    Update parser.mly

commit f4d2e18c5501a34dfbdedbe712f8a71d8f50acc0
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Sun Nov 21 13:24:40 2021 -0500

    Add files via upload

commit 1bc580e227b5494e83cecad4473009799429c905
Author: Jeremy Lu <jeremylu04@gmail.com>
Date:   Sun Nov 21 12:57:52 2021 -0500

    Update README.md

commit a4b3471735637dc997c3319d133cbf32ec935b6e
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Sun Nov 21 12:35:13 2021 -0500

    Add files via upload

commit b0f1a468996a84b7da8dd6928c9de4a1962ca310
Author: connieeez <51053428+connieeez@users.noreply.github.com>
Date:   Fri Nov 19 15:17:18 2021 -0500

    Add files via upload

    microc.ml added

commit 758ed2b1f62f9074d8249c081e60f1d7e7819318
Author: ebschwartz3 <71985184+ebschwartz3@users.noreply.github.com>
Date:   Thu Nov 18 12:56:36 2021 -0500
```

```
    Create Dockerfile

commit 7ec60ae49377d10032cabd43b6c7b019992a1f01
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Tue Nov 16 19:21:12 2021 -0500

    Update ast.mli

commit b40601945290aeb76d6297cbc23b66c015be9804
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Tue Nov 16 19:20:41 2021 -0500

    Update ast.mli

    condensed ast

commit ca85cfcbecd699d57671009ca7e0568cd4b17456
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Tue Nov 16 19:20:05 2021 -0500

    Update parser.mly

    condensed

commit d96bca7f3134a062f3eb8d0823aa24c5ebd9f7cd
Author: lujeremy <jeremylu04@gmail.com>
Date:   Tue Nov 16 11:52:37 2021 -0500

    Rehash makefile, add tags, rename proj

commit c03f7a1b6a1546d9bd86ed14501250f1ba60b136
Author: Jeremy Lu <jeremylu04@gmail.com>
Date:   Thu Nov 11 13:54:53 2021 -0500

    Update README.md

commit 4e2930d3a7e5f5c6cf0a414005793262f45e797c
Author: lujeremy <jeremylu04@gmail.com>
Date:   Thu Nov 11 13:18:01 2021 -0500

    Add c program to draw circle
```

```
commit 39c294e96372f791c573da62eddda17df3d611d6
Author: lujeremy <jeremylu04@gmail.com>
Date:   Thu Nov 11 11:25:04 2021 -0500

    Add test file

commit f04710f600d7e905ce645a83ef5f967bfbddb821
Author: lujeremy <jeremylu04@gmail.com>
Date:   Thu Nov 11 11:23:42 2021 -0500

    Add makefile

commit 8b205de1af3dbaec7e056b87c7b37e9898f2a39d
Author: melodyhsu47 <36453835+melodyhsu47@users.noreply.github.com>
Date:   Sun Nov 7 12:40:21 2021 -0500

    Add files via upload

commit f7aa736a86a840ea4e159d07fab81c19049e788a
Author: lujeremy <jeremylu04@gmail.com>
Date:   Sun Nov 7 09:28:38 2021 -0500

    Add calc.ml as template for our parser

commit 441937345454c6d4de5e6b250bb1f390b703c110
Author: Jeremy Lu <jeremylu04@gmail.com>
Date:   Tue Oct 26 18:23:44 2021 -0400

    Initial commit
```