SSOL

Simple Shape Oriented Language

Madeleine Tipp	Jeevan Farias	Daniel Mesko
mrt2148	jtf2126	dpm2153
Manager / Test Architect	Language Designer	System Architect

December 19, 2018

Contents

Introduction

SSOL is a programming language that allows users to create shapes algorithmically and render them in an SVG file. It features two built-in shape objects, Point and Curve, which can be used as building blocks to define more complex polygons or curved figures. The shapes are then added to a user-defined Canvas object, which abstractly represents the plane on which the shapes are to be drawn. The Canvas object can then be passed into the built-in *draw*() function to be rendered and stored as an SVG file. Without using *draw*(), SSOL functions as a minimal, general purpose programming language similar to C.

Chapter 1

Language Tutorial

1.1 Requirements

package	version
opam	1.2.2
llvm	6.0.0
ocamlfind	1.8.0

1.2 Compiler

1.2.1 Building the Compiler

Run `make` to build the compiler excecutable, `ssol.native`.

1.2.2 Running the Compiler

Run `./ssol <target_ssol_filename>` to compile your SSOL source code and run the resultant executable. Note that the target file should be specified *without* the *.ssol* extension. Also note that the executable generated from your SSOL code is deleted.

```
Ex. ./ssol mytest
```

1.2.3 Output files

Output files will be written to whatever file name is specified with the program logic. To view these output SVG images, open the desired file in a web browser, or any image editing program compatible with the SVG fileformat.

1.3 Code Walkthrough

Below is the complete logic for a program that produces a square.

1.3.1 Full Sample Source

Every program in SSOL must have a main method defined by the programmer. This method returns int.

```
int main(){

    float l = 1000.0;
    float w = 1000.0;

    Canvas can = Canvas(l,w);
```

```
    //Create 4 straight lines that form a square
    Point up_lc = Point(w*0.1,l*0.1);
    Point up_rc = Point(w*0.9,l*0.1);
    Point bt_lc = Point(w*0.1,l*0.9);
    Point bt_rc = Point(w*0.9,l*0.9);

    Curve top = Curve(up_lc, up_rc, up_lc, up_rc);
    Curve right = Curve(up_rc, bt_rc, up_rc, bt_rc);
    Curve bottom = Curve(bt_lc, bt_rc, bt_lc, bt_rc);
    Curve left = Curve(up_lc, bt_lc, up_lc, bt_lc );

    can |= top;
    can |= right;
    can |= bottom;
    can |= left;

    draw(can, "output.svg");
    return 0;
}
```

1.3.2 Annotated Code Fragments

Declare our length as widh as floats, instantiate our canvas with these dimensions.

```
    float l = 1000.0;
    float w = 1000.0;

    Canvas can = Canvas(l,w);
```

Create our four points that will be the corners of the square. We specify their coordinations in terms of the length and width, declared above.

```
    Point up_lc = Point(w*0.1,l*0.1);
    Point up_rc = Point(w*0.9,l*0.1);
    Point bt_lc = Point(w*0.1,l*0.9);
    Point bt_rc = Point(w*0.9,l*0.9);
```

Instantiate the four curves that will create the lines of the square. *Curve*() takes four points, two end points and two control points. If we use the end points as the control points, the resulting Bezier curve is a straight line.

```
    Curve top = Curve(up_lc, up_rc, up_lc, up_rc);
    Curve right = Curve(up_rc, bt_rc, up_rc, bt_rc);
    Curve bottom = Curve(bt_lc, bt_rc, bt_lc, bt_rc);
    Curve left = Curve(up_lc, bt_lc, up_lc, bt_lc );
```

Add the curves to the canvas using | = (pipend). This operator can take a *Canvas* on the left and a *Curve* on the right. Sequential addition can happen as follows, or within a loop from an array for example.

```
    can |= top;
    can |= right;
    can |= bottom;
    can |= left;
```

Finally, *draw*() the canvas, which will translate all the curves that have been added to their SVG representations, and write the subsequent file to the destination specified. The file is written to the directory in which you run *./ssol*. We return 0, to signify program exit, and match the return type of *main*

```
    draw(can, "output.svg");
    return 0;
```

Chapter 2

Language Reference Manual

2.1 Lexical Conventions

2.1.1 Identifiers

Identifiers consist of one of more characters where the leading character is a uppercase or lowercase letter followed by a sequence uppercase/lowercase letters, digits and possibly underscores. Identifiers are primarily used in variable declaration.

2.1.2 Keywords

Keyword	Definition
if	initiates a typical if-else control flow statement
else	
while	initiates a while loop
for	initiates a for loop
return	returns the accompanying value (must be of the appropriate return type)
void	used to identify a function that does not return a value
int	type identifier for int
float	type identifier for float
bool	type identifier for bool
char	type identifier for char
String	type identifier for String
Point	type identifier for Point
Curve	type identifier for Curve
Canvas	type identifier for Canvas
true	literal Boolean value
false	literal Boolean value

2.1.3 Literals

Type	Definition
Int	A sequence of one or more digits representing an un-named(not associated with any identifier) integer, with the leading digit being non-zero (i.e. [1-9][0-9]*)
Float	A sequence of digits seperated by a '.' representing an un-named float-point number (i.e. [0-9]*.[0- 9][0- 9]*)
Char	A single character enclosed by single quotation marks representing an un-names character. (i.e. '.')
String	A sequence of characters enclosed by a pair of double quotation marks representing an un-named string. (i.e. ^ ".*" $)
Bool	8-bit boolean variable, either $true$ or $false$

2.1.4 Comments

SSOL supports single line and multi-line comments. Single line comments are initiated by two "/" characters. (i.e. //), and are terminated by a newline character. Multi-line comments are initiated by the character sequence '/*' and terminated by the character sequence '*/*.

```
// This is a single line comment

/* This is a
     multi-line comment */
```

2.1.5 Punctuators

A punctuator is a symbol that has semantic significance but does not specify an operation to be performed. The punctuators [], (), and {} must occur in pairs, possibly separated by expressions, declarations, or statements. The semi-colon (;) is used to denote the end of every statement or expression. SSOL includes the following punctuators: [](),;

SSOL uses the semi-colon for sequencing and denoting the end of an operation. Terminate every statement with a semicolon (;).

2.1.6 Whitespace

Whitespace (space, tabs, and newlines) is ignored in SSOL.

2.2 Types

2.2.1 Primitives

Type	Definition
int	4 byte signed integer
float	8 byte floating-point decimal number
bool	1 byte Boolean value
char	1 byte ASCII character
String	array of ASCII characters

2.2.2 Complex Types

The following built-in complex data types are represented as objects with member fields and are instantiated using their associated constructors. The individual fields of the objects can be accessed and modified with . notation, ex: `object.field`

Point

A Point object contains two fields: an x and a y coordinate value, both of type `float`.
A Point object is instantiated using the following constructor:

```
Point(float x, float y)
```

Curve

A Curve object represents a Bezier curve, defined by two endpoints and two control points. Curves are instantiated using the following constructor.

```
Curve(Point ep1, Point ep2, Point cp1, Point cp2)
```

The constructor creates a Curve defined by endpoints a and b and control points $c1$ and $c2$. This constructor will accept Point identifiers or Point constructors.

Canvas

A Canvas object represents a two-dimensional coordinate plane to which Point and Curve objects are added. These graphical elements are added using the | = operator. Canvas objects are outputted to files via the *draw* library function.

A Canvas object is instantiated using the following constructor:

```
Canvas(float x, float y)
```

This constructor creates a Canvas object with the dimensions specified by the values for x and y.

2.2.3 Arrays

Arrays are a built-in data structure consisting of sequential elements of a single type. See section 2.3.2 for usage.

2.3 Syntax

2.3.1 Type Specifiers

SSOL is a language with explicit typing. All variables and functions must be declared with a type specifier, which tells the compiler which operations are valid for the former and what to expect the latter to return. Declaration and assignment can happen in seperate or combined statements.

Primitives

```
int x;
x = 3;
String myString = "hello";
bool b = true;
```

Complex Types

Below are usage examples of our complex types: Curve, Point, and Canvas.

```
//Canvas can be instantiated with default size or with a user specified size
Canvas can2 = Canvas(100.0,100.0);

Point pt = Point(10.0,20.0);

/* Create a straight line by declaring a Curve
 * using the end points as the control points
*/
```

```
Curve crv1 = Curve( Point(10.0,20.0), Point(100,200), Point(10.0,20.0), Point(100,200) );

/* Create a bezier curve by declaring a Curve using 4 arguments
 * Here we demonstrate that Curve will except
 * any combination of valid arguments.
*/
Curve crv2 = Curve( pt, Point(40.0,40.0), Point (100.0,100.0), pt );
```

2.3.2 Arrays

Arrays in SSOL are instantiated with a fixed size and can only hold a single type, which can be either primitive or complex. Declaration can happen as a statement containing declaration with a fixed size, or declaration and assignment to an array literal. Size must be specified as an *int* literal.

Declaration

```
int arr[5];
arr = {1,2,3,4,5};

int arr2 = {1,2,3,4,5};
```

Accessing

Use brackets and an index to retrieve a value from an array. The specified index must be within the bound of the array. The variable returned by the array access operation must match the variable that its value is assigned to.

```
float f = floatArr[0];
Point p = Point(floatArr[0], floatArr[1]);
```

2.3.3 Operators

Arithmetic

Addition (+), subtraction (-), multiplication (*), division (/), and modulo (%) are standard arithmetic operators in SSOL which comply with order of operations. These operators are valid for *int* and *float* independently, but cannot be used on *int* and *float* together.

Comparison

Comparison in SSOL is done via ==, !=, <,>,<=, and >=. Only matching types can be compared. These operators return a Boolean value of *true* or *false*

Logical

SSOL can perform logical operations of Boolean values with `&&` (AND), || (OR), and ! (NOT).

```
bool b1; b1 = true && true;
bool b2; b2 = false || false;
```

Assignment

The assignment statement is of the form `<identifier> = expression`, where `<identifier>` has been previously declared.

Canvas

The Canvas object of SSOL has a unique operators for sequencing and addition to a the canvas. Only Curve objects can be added to a Canvas, and must be added one at a time.

|=	Use pipend to add a Curve to a canvas

Ex.

```
canvas |= crv1;
canvas |= crv2;
canvas |= crv3;
canvas |= crv4;
```

2.3.4 Statements

Sequencing

Consecutive statements are sequenced using the `;` operator.

Control Flow

SSOL supports the standard `if...else` format of conditional statements. `if` requires a Boolean statement to be evaluated.

```
int i = 3;
if(i>4){
    print("i > 4");
} else {
    if(i<3){
        print("i < 3");
    }
    else{
        print("i == 4");
    }
}
```

Loops

SSOL supports **for** loops and **while** loops. For loops are an iterative construct that requires a starting index variable, a bounding condition, and an operation to be performed at the end of each iteration.
A **while** loop requires a Boolean expression to be evaluated every time the loop is executed.

```
int i;
for(i = 0; i<arr.length; i++){
    <loop-body>
}

int j; j = 0;
while(j<10){
    j+=1;
}
```

2.3.5 Functions

Declaration

Functions are declared as follows:

```
<function-return-type> <function-name>([arg1],[arg2],...)
{
    <function-body>
    [return <some-value>]
}
```

If the function has non-void return type, then it must return some value of that type at the end of the function, or at the end of any potential path of execution within the function, if there are conditional statements/loops. This is achieved using the keyword `return`.

Function Calls

Functions are called as follows:

```
<function-name>([arg1],[arg2],...)
```

If a function returns a value, that value can be assigned to a variable, assuming the variable has been previously declared, as in

```
<identifier> = <function-name>([arg1],[arg2],...)
```

2.4 Execution

2.4.1 Scope

Variables persist only within the block of code in which they are declared. A block of code is enclosed by curly braces ({ ... }). Variables that are declared outside of any code block are considered global and are visible to all functions within a program.

2.4.2 main()

Every valid SSOL program needs at least one function called *main*(). This is the routine that will be executed at runtime, so program trajectory must start from here. Within *main*(), other user-defined functions may be called. The return type of *main*() is *int*.

2.5 Built-In Functions

2.5.1 draw()

draw() is the crux of the SSOL language. This method takes a single *canvas* object and a file name as a string as arguments. *draw*() can be called as many times as the programmer desires, but there will be a 1:1 correlation between function calls and .SVG files written (if *draw* is called with the same filename twice, the file will be overwritten).

```
Ex. draw(<Canvas>, <String>");
```

2.5.2 printf()

printf() is function to print *float* values.

```
Ex. printf(<float>);
```

2.5.3 print()

print() is function to print *string* values.

```
Ex. print(<int>);
```

2.5.4 printbig()

printbig() is function to print ascii values.

```
Ex. printbig(<int>);
```

2.5.5 prints()

prints() is function to print *string* values.

```
Ex. prints(<String>);
```

2.6 Sample Code

```
int main(){

    float l = 1000.0;
    float w = 1000.0;

    Canvas can = Canvas(l,w);

    //Create 4 straight lines that form a square
    Point up_lc = Point(w*0.1,l*0.1);
    Point up_rc = Point(w*0.9,l*0.1);
    Point bt_lc = Point(w*0.1,l*0.9);
    Point bt_rc = Point(w*0.9,l*0.9);

    Curve top = Curve(up_lc, up_rc, up_lc, up_rc);
    Curve right = Curve(up_rc, bt_rc, up_rc, bt_rc);
    Curve bottom = Curve(bt_lc, bt_rc, bt_lc, bt_rc);
    Curve left = Curve(up_lc, bt_lc, up_lc, bt_lc );

    can |= top;
    can |= right;
    can |= bottom;
    can |= left;

    draw(can, "output.svg");
}
```

Chapter 3

Project Plan

3.1 Development Process

3.1.1 Planning

Our team met weekly on Wednesday evenings, immediately following our weekly meeting time with T.A. Mark Mazel. This proved very productive, as we could ask Mark any questions we had about the development process, and then immediately work to resolve them based on his feedback. In anticipation of deadlines, we also met on Fridays, Saturdays, and Sundays, based on member availability. The team used a Slack workspace to communicate about meeting times, issues related to the work of other team members, and larger version control concerns. GitHub (where we hosted our code repository) was integrated into the Slack workspace so that all pull requests and commits to the master branch were broadcast to all members of the channel. This helped to make everyone aware when they needed to update their local copies of the code.

3.1.2 Specification and Development

During our meetings, we would typically plan out architectural aspects of the compiler, such as specifying the evolution of a particular expression or statement as it passes through parsing, semantic checking, and code generation, and what needs to happen to it at each stage. We would then each set goals for ourselves to work towards before our next meeting, based on the architectural decisions made, with respect to each individuals strengths and roles. While we did not always meet these goals, having them set each week helped us to identify the major tasks in the various phases of the compiler development, so that we were not blindsided by a task just before the deadline. If certain weekly goals proved ambitious, the team came together at the next session to break them up into smaller goals and split the work load more appropriately, or in some cases, to work collectively on the problem.

3.1.3 Testing

Most of our end-to-end testing took place during the final weeks of the semester, as it was difficult to test the compiler without having the code generation phase finished. We were however able to test elements of the semantic checking before code generation was complete by running the compiler with the `-s` flag to see the `SAST` printed to the console. Since we usually split up the work by particular expressions or statements (or groups of expressions or statements), we delegated the responsibility of writing tests for those patterns to the individual who implemented them. In some cases, where the semantic checking of a pattern was delegated to one team member, and the code generation to another, the two team members worked together either in person or over the Slack channel to write tests and resolve errors. Our Test Architect Madeleine modified the test suite to convert legacy tests from MicroC to match SSOL syntax, and have the `.ssol` extension.

3.2 Programming Style and Conventions

The team used the following conventions to maintain consistent code structure and readability:

- Lines should not exceed 80 characters.
- Spaces should be used instead of tabs, so that different editors do not render them differently.
- Nested code should be indented by two spaces in OCaml, and four spaces in C.
- Function names should be lowercase snake_case and variable names in camelCase.
- Recursively evaluated expressions and statements, if bound locally, should take the name of the original argument, with an apostrophe added (i.e. `let x' = expr locals x`).
- Any functions added should include a comment above the signature describing what the function does, and the context(s) in which it would be called.
- Function calls, extended string concatenations, or lines that otherwise extend past 80 characters should be broken up into multiple lines, and those lines indented four spaces beyond the first line, to make clear that those lines are part of one expression, and not a separate, nested expression.

3.3 Project Timeline

September 16	Creation of code repository, sample program and initial commit
September 19	Submitted Project Proposal
October 15	Submitted Scanner, Parser, and Language Reference Manual
November 14	Submitted "Hello World" program
December 1	Completed Semantic Checking
December 10	Completed Code Generation
December 18	Presented project and Fibonacci spiral program
December 19	Submitted report and code

3.4 Roles and Responsibilities

Team Member	**Role**
Madeleine Tipp	Manager and Test Architect
Jeevan Farias	Language Designer
Daniel Mesko	System Architect

While each team member had a specific role, we all worked collectively on all aspects of the compiler, test suite, and report. We did however break up the individual work based on our roles. For example, Jeevan wrote sample programs, since he was Language Designer. Test Architect Maddy designed most of the end-to-end tests and updated our test script. System Architect Daniel wrote the C code for the SVG file generation and updated the semantic checking and code generation for built-in functions to link it in. The Parser and Scanner were written collectively. Semantic checking of expressions, semantic checking of statements, and code generation for statements were divided among the team members. Code generation for expressions was completed collectively.

3.5 Development Environment

The following tools and languages were used in development:

- **Git/GitHub** - for version control, our team used **git** in the command line, linked with a remote code-repository hosted by **GitHub**. When team members wanted to implement a feature, they branched off of the master branch, worked within that branch until the feature was implemented and tested and the compiler built without errors, and then created a "Pull Request" on **GitHub** to merge those changes back into the master branch. Pull Requests were carefully reviewed by all members before being merged into master, and any merge conflicts that arose were settled in the command line with all members present before merging.
- **Slack** - the team used the Slack collaboration tool, integrated with GitHub, to communicate with one another about logistics and development issues and to remain updated as to the state of the master GitHub branch.
- **Ubuntu Virtual Machine** - the team developed within the "numel" Ubuntu GNU/Linux virtual machine provided by T.A. John Hui to ensure that the necessary OCaml/Opam and LLVM dependencies were installed and the versions consistent between team members.
- **OCaml toplevel, version 4.05.0** - used to compile the semantic checker (`semant.ml`), the code generator (`codegen.ml`), and the top-level phase linker (`ssol.ml`).
- **OCamllex, version 4.05.0** - used to generate the SSOL scanner/lexer (`scaner.mll`)
- **OCamlyacc, version 4.05.0** - used to generate the SSOL parser (`parser.mly`)
- **OCamlbuild, version 0.12.0** - invoked by Makefile to automate the building process of the various compiler source files.
- **LLVM, version 6.0.0.0** - used to generate the LLVM IR output by our compiler executable (`ssol.native`), and to compile the LLVM IR into native assembly code.
- **GCC, version 7.3.0** - used to compile our linked C code `svg.c, draw.c, printbig.o` and to link the assembly code generated from the LLVM IR with the relevant C object code.
- **GNU Make** - used to build our compiler and its dependencies, as well as run our test script.

3.6 Project Log

```
commit f05d8ebc49a9a544cc8576769b5729dcf28adefe
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Wed Dec 19 23:49:02 2018 -0500

    Added tests to tarball dependency

commit a137f47460cdf219363cd4ce6a3092d9252b8429
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Wed Dec 19 22:53:15 2018 -0500

    Added array and field fail tests

commit fb20513e52786d450d4e59a969afc31e7a839c0f
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Wed Dec 19 22:52:53 2018 -0500

    cleaned up indentation

commit c0dbadf4c18bd2cfcc72106f79848df83c164c63
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Wed Dec 19 22:04:00 2018 -0500

    spacing in semant, more array error tests

commit 8b298b3b87d688694703e1411eb11c2ce80db385
Author: mtipp <mrt2148@barnard.edu>
Date:   Wed Dec 19 18:37:46 2018 -0500

    Update field tests and delete /temptests.
```

```
30
31 commit 69cd7360a85c8b653b8b4b594e68d68aa8d99ca8
32 Author: mtipp <mrt2148@barnard.edu>
33 Date:   Wed Dec 19 18:14:12 2018 -0500
34
35     Update array tests. fails oob.
36
37 commit c6bafd463d877e88525e02df32ad5f2744dd9209
38 Author: Daniel Mesko <dpmesko@gmail.com>
39 Date:   Wed Dec 19 17:09:46 2018 -0500
40
41     Added authorship in file headers
42
43 commit ac48c81db563d2ae4218eb7eeb3e6253d7fd588f
44 Author: jvntf <jeevanfarias@gmail.com>
45 Date:   Wed Dec 19 15:44:23 2018 -0500
46
47     Testfailures (#39)
48
49     * updated testall,added pipend test
50
51 commit a7ed1734f548d83bc5b688f7d312b899055bcc2c
52 Merge: 6afae6f f7211dc
53 Author: Daniel Mesko <dpmesko@gmail.com>
54 Date:   Wed Dec 19 15:26:00 2018 -0500
55
56     updating local copy of master - Merge branch 'master' of https://github.com/
         dpmesko/ssol
57
58 commit 6afae6fd9e37d843f410864fa68f8885fa56fdf9
59 Author: Daniel Mesko <dpmesko@gmail.com>
60 Date:   Wed Dec 19 15:25:53 2018 -0500
61
62     Fixed spacing
63
64 commit f7211dc965b8c27bedc74d48b341a971ac0cd172
65 Author: jvntf <jeevanfarias@gmail.com>
66 Date:   Wed Dec 19 15:22:32 2018 -0500
67
68     Testfailures (#38)
69
70     * test updates. still failures on makefile printbig.o
71
72     * Update tests and err messages.
73
74     * fixed testall script
75
76     * all tests pass
77
78 commit 03335e83b9875f70bbb9dd0469d2bd93c40d4289
79 Author: jvntf <jeevanfarias@gmail.com>
80 Date:   Wed Dec 19 13:54:13 2018 -0500
81
82     added a prints function to print strings (#37)
83
84 commit 01d41685bf23981ce84f24ad4b0fc4684e163a3d
85 Merge: 9ea81af a3900fe
86 Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
87 Date:   Tue Dec 18 21:36:15 2018 -0500
88
89     Merge pull request #36 from dpmesko/cleanup
90
91     Replaced tabs with spaces in OCaml code, cleaned up indentation, remo
92
93 commit a3900fe3929f8335e2c4ab43ed16a749ea6b83b6
94 Author: Daniel Mesko <dpmesko@gmail.com>
95 Date:   Tue Dec 18 21:35:29 2018 -0500
96
97     Replaced tabs with spaces in OCaml code, cleaned up indentation, removed old
         comments - generally B E A U T I F I E D
98
99 commit 9ea81afc089f06ce4d0989b0944ea259d17bd56c
```

```
Merge: d7134a4 28c18b0
Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
Date:   Tue Dec 18 16:45:54 2018 -0500

    Merge pull request #35 from dpmesko/cleanup

    merging cleanup branch

commit 28c18b024c868c2add98ea0c7c57a6f2ba7d0c32
Merge: 569a809 d7134a4
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Tue Dec 18 16:44:58 2018 -0500

    merged master, fixed conflicts

commit 569a809401966d87e9673576187a0ce700ca226d
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Tue Dec 18 16:36:33 2018 -0500

    temporarily commented out testall target from Makefile

commit 50f2c6899c93c37213611a329445c39dd59d4803
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Tue Dec 18 16:35:09 2018 -0500

    removed unnecessary comment

commit d7134a4d0bffd6bdd0ae366b5907273f35f697de
Author: jvntf <jeevanfarias@gmail.com>
Date:   Tue Dec 18 01:41:15 2018 -0500

    Demo program (#34)

    * finished demo program

commit 8bbb4d9a37b1dddbb88df4219828ac0d702f5eca
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Tue Dec 18 00:07:39 2018 -0500

    No more compiler warnings! Unused variables removed, unmatched patterns caught
        with wildcards

commit 348cc6089e7696945ce4e92e9dcf3d8ec0be90fd
Author: jvntf <jeevanfarias@gmail.com>
Date:   Mon Dec 17 22:54:33 2018 -0500

    fixed fib spiral file (#33)

    * fixed fib spiral file

    * put back the SCall that went missing again

commit 1abeeaf0505068f10f9152236c939cb3a847edcf
Merge: c209552 a864d9b
Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
Date:   Mon Dec 17 21:54:27 2018 -0500

    Merge pull request #32 from dpmesko/pipend-dan

    Implemented pipend, removed pipe, removed Constructor in favor of Call, some
        cleanup

commit a864d9b8273720622d246695c6d9ff6a9a846634
Merge: 5afc149 c209552
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Mon Dec 17 21:36:52 2018 -0500

    Merging with master

commit 5afc149cb32feb0640f188bab1278ee602285e9a
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Mon Dec 17 21:32:32 2018 -0500
```

```
170
171     Removed unnecessary files from compiler directory
172
173 commit c16ebca7cac43db3682fa74b47685d702ae8e85a
174 Author: Daniel Mesko <dpmesko@gmail.com>
175 Date:   Mon Dec 17 21:26:56 2018 -0500
176
177     Converted Constructor/SConstructor to Call/SCall so that we can check args to
        constructors in semant
178
179 commit c209552d6c0e56046955d9956b4ab52c5d6dcf79
180 Author: jvntf <jeevanfarias@gmail.com>
181 Date:   Mon Dec 17 20:37:11 2018 -0500
182
183     Revert "augmented print call to print anything. identifiers print as 'Identifier:
        %name' (#30)" (#31)
184
185     This reverts commit 58b6b73ae0226649dcc41b81cf2f39e6e6d89aa1.
186
187 commit 58b6b73ae0226649dcc41b81cf2f39e6e6d89aa1
188 Author: jvntf <jeevanfarias@gmail.com>
189 Date:   Mon Dec 17 20:23:40 2018 -0500
190
191     augmented print call to print anything. identifiers print as 'Identifier: %name'
        (#30)
192
193 commit df8139a36f4620a06d4cd31d4bef17c258691759
194 Author: Daniel Mesko <dpmesko@gmail.com>
195 Date:   Mon Dec 17 16:38:37 2018 -0500
196
197     removed pipe operator from all parts of compiler - no longer a feature of SSOL
198
199 commit 100e41ed71ed5035f1cd782c6c514c6fd7030b4c
200 Author: Daniel Mesko <dpmesko@gmail.com>
201 Date:   Mon Dec 17 03:12:11 2018 -0500
202
203     modified test-draw
204
205 commit 080b5368e12c9febd4e5ff1ce0bdfb15bcb49999
206 Author: Daniel Mesko <dpmesko@gmail.com>
207 Date:   Mon Dec 17 02:56:04 2018 -0500
208
209     Removed unecessary arg to Canvas constructor, but still returns garbage from
        linked C code. Cleane dup Makefile target structure
210
211 commit 0f4fbe06bcbe29826acc7ba3be05340a954772db
212 Author: jvntf <jeevanfarias@gmail.com>
213 Date:   Mon Dec 17 01:21:59 2018 -0500
214
215     Control flow tests (#29)
216
217     * got rid of some unused variable warnings
218
219     * added codegen:expr->SArrayAssign
220
221     * added charlit
222
223     * added user defined function calls
224
225     * reversed arg checking in codegen -> SCall
226
227     * matched some patterns to get rid of warnings
228
229     * Change test extensions and testall.sh from .mc to .ssol.
230
231     * updated ssol script to not remove files if u accidentally run with the file
        extension
232
233     * updated tests and fixed output files
234
235 commit dfce6af19ee0fa457742af173a539dbd7477244f
236 Author: Daniel Mesko <dpmesko@gmail.com>
```

```
Date:   Mon Dec 17 00:46:39 2018 -0500

    some changes to canvas constructor, draw()...trying to fix double/float issue

commit 32bb8490554465ac5c3d8eed2e2f1bfb031f3831
Author: jvntf <jeevanfarias@gmail.com>
Date:   Mon Dec 17 00:33:06 2018 -0500

    Function returns (#28)

    Function return types

commit cf988518c8542c70ad301e2e1e652bc8e505a779
Author: jvntf <jeevanfarias@gmail.com>
Date:   Sun Dec 16 23:57:37 2018 -0500

    Pipend dan (#26)

    * removing files that should have never been comitted

    * Start work on pipend.

    * Small error fix, need to declare helper fcn I think.

    * Pipend compiles but segfaults test.

    * Fixed weird float/double inconsistency in SVG code with Canvas construction,
      draw calls

    * [WIP] Pipend codegen, canvas pointer manipulation

    * Got pipend to work for multipe curves

    * A little draw test that makes three curves and adds them to Canvas, then calls
      draw()

commit 1150c69597554d255ba4046de1c860810694acba
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Sun Dec 16 23:43:49 2018 -0500

    A little draw test that makes three curves and adds them to Canvas, then calls
      draw()

commit 416e3a0720a4986751c396462fca7c36fc583b59
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Sun Dec 16 23:43:24 2018 -0500

    Got pipend to work for multipe curves

commit 7e1bc613d9b956c6b32839309047566e42cd975b
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Sun Dec 16 23:26:35 2018 -0500

    [WIP] Pipend codegen, canvas pointer manipulation

commit 17f505df718f0f2ad3b6f5da7c1e3f8ab3059bf1
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Sun Dec 16 21:13:03 2018 -0500

    Fixed weird float/double inconsistency in SVG code with Canvas construction, draw
      calls

commit 94de4523c1eb3a29d3ee3101b7b27dc96d642a89
Author: jvntf <jeevanfarias@gmail.com>
Date:   Sun Dec 16 20:34:11 2018 -0500

    Constructor vars (#25)

    * tests to fix construct var thig, commented out main in draw,c

    * Constructors use build call to create structs
```

```
commit 912fd802baaa9d6410b41712d0409dce34db97b4
Merge: 05dea35 ddd2337
Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
Date:   Sun Dec 16 20:31:54 2018 -0500

    Merge pull request #24 from dpmesko/draw-link

    draw() now takes Canvas objects, links with SVG code

commit 08c16ff089675fa889a50b57693e2d603a86859b
Merge: 1ca75ba 94de452
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Sun Dec 16 20:29:54 2018 -0500

    just do it alreadyMerge branch 'master' into codegen-maddy

commit ddd2337f3ccb88e132ca2ebc58f022db25d1f14a
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Sun Dec 16 20:05:22 2018 -0500

    draw() now takes Canvas objects, links with SVG code

commit 05dea356f8400043eaaf33c5e353641ce73fb5d9
Merge: 6a0a1f7 5e263d9
Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
Date:   Sun Dec 16 16:52:53 2018 -0500

    Merge pull request #22 from dpmesko/canvas-assign

    Fixed error with Canvas construction - Lconst_pointer_null is garbage

commit 1ca75baf400702e23aa0992b4f9b272d4dfdabc9
Merge: 584e1ff 05dea35
Author: mtipp <mrt2148@barnard.edu>
Date:   Sun Dec 16 16:44:24 2018 -0500

    Merge branch 'master' into codegen-maddy

commit 5e263d9cf894ba33960ab2a5f10a309144e8f632
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Sun Dec 16 16:41:15 2018 -0500

    Fixed error with Canvas construction - Lconst_pointer_null is garbage, L.
        const_null (L.pointer_type) is better; commented out conflicting main in draw.
        c, updated tests

commit 584e1ff9c55040d614c837ddd4c83966d8fa1600
Author: mtipp <mrt2148@barnard.edu>
Date:   Sun Dec 16 16:39:57 2018 -0500

    Pipend compiles but segfaults test.

commit 6a0a1f7a08be978d85ac77d5910417fcb9f8a89e
Merge: 433e8bd 13f58e5
Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
Date:   Sun Dec 16 16:03:09 2018 -0500

    Merge pull request #21 from dpmesko/svg-draw

    SVG linking (attempted)

commit 13f58e5d7ff1145336b2e3bf202e08ce31f5090b
Merge: 52965b5 433e8bd
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Sun Dec 16 16:01:49 2018 -0500

    Merging master

commit 52965b51361edfb7483617505fb7fbcb752e37d2
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Sun Dec 16 15:58:05 2018 -0500
```

```
375       [WIP] Trying to figure out Canvas struct problems
376
377   commit 433e8bd91fbd0a39d68d6135adaf82cbe47c531e
378   Author: jvntf <jeevanfarias@gmail.com>
379   Date:   Sun Dec 16 15:57:09 2018 -0500
380
381       Codegen expr (#20)
382
383       * got rid of some unused variable warnings
384
385       * added codegen:expr->SArrayAssign
386
387       * added charlit
388
389       * added user defined function calls
390
391       * reversed arg checking in codegen -> SCall
392
393       * matched some patterns to get rid of warnings
394
395       * field going in the right direction, not working yet
396
397       * codegen SField working and compiling, assignment will throw semantic error when
              doing field assignment to variable
398
399       * got field codegene working, but problems with assignment to a variable
400
401       * changed order of arguments for Canvas constructor to match the others
402
403       * field assignment working...semantic errors get thrown at runtime instead of
              compile time
404
405   commit a083461f0794514eb455de2d68e65ea1a6583be9
406   Author: Daniel Mesko <dpmesko@gmail.com>
407   Date:   Sun Dec 16 15:37:49 2018 -0500
408
409       [WIP] fixing draw linking
410
411   commit 7159bc2181ae5c4782f85102eddee4e9738ac733
412   Merge: 53a26b3 23fbe3d
413   Author: mtipp <mrt2148@barnard.edu>
414   Date:   Sun Dec 16 05:09:20 2018 -0500
415
416       Merge branch 'codegen-maddy' of https://github.com/dpmesko/ssol into codegen-maddy
417
418   commit 53a26b3fedb7ef6d1c616efa999f3ea7905fa5fa
419   Author: mtipp <mrt2148@barnard.edu>
420   Date:   Sun Dec 16 05:08:58 2018 -0500
421
422       Small error fix, need to declare helper fcn I think.
423
424   commit 841c3d17b800002bf0d9d3b6b619c13ad6d7678a
425   Author: mtipp <mrt2148@barnard.edu>
426   Date:   Sun Dec 16 05:00:34 2018 -0500
427
428       Start work on pipend.
429
430   commit 82952eabf3e79a8d6bf21fc8400ae7f73158327f
431   Author: Daniel Mesko <dpmesko@gmail.com>
432   Date:   Sun Dec 16 04:02:42 2018 -0500
433
434       Updated SVG code to draw bezier cues, walk the canvas_nodes, added test main
              function and Makefile target
435
436   commit 830f1b1bf59d8b16178b074434990eabbc9a5d59
437   Merge: 35289cd 2583f1f
438   Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
439   Date:   Sun Dec 16 01:31:00 2018 -0500
440
441       Merge pull request #19 from dpmesko/svg-draw
442
443       semant, codegen, SVG lib updated for new definition of draw(), codegen updated for
```

```
          new canvas_node definition
444
445 commit 2583f1f1ddd5b35296911b4eaea3eadd186efd9b
446 Author: Daniel Mesko <dpmesko@gmail.com>
447 Date:   Sat Dec 15 23:22:14 2018 -0500
448
449     Updated args to draw() in semant and codegen, updated pipe/pipend in semant to
          check for Curves only, consistent with new canvas_node definition, added draw
          test file (does not work yet), removed unnecessary test files
450
451 commit 23fbe3d0a7ac664058322e355d9cd5b4ddc806a8
452 Author: Daniel Mesko <dpmesko@gmail.com>
453 Date:   Sat Dec 15 22:56:36 2018 -0500
454
455     removing files that should have never been comitted
456
457 commit e237876fdb6b0ce68d544cd96327a769af4a58ef
458 Merge: e5733a8 c0f1cd2
459 Author: Daniel Mesko <dpmesko@gmail.com>
460 Date:   Sat Dec 15 22:47:36 2018 -0500
461
462     merging in codegen-maddy, fixing conflicts
463
464 commit e5733a825e274f447d7414c2d1f5bc978ac56585
465 Author: Daniel Mesko <dpmesko@gmail.com>
466 Date:   Sat Dec 15 22:43:17 2018 -0500
467
468     Changed curve struct definition - Canvas objects will no longer reference Points,
          because they're sorta not renderable by themselves, so they're not needed by
          the SVG code that gets passed a canvas pointer
469
470 commit 22ca73d5aa82d7bf7fad68ceada0d0de02b9a79c
471 Author: Daniel Mesko <dpmesko@gmail.com>
472 Date:   Sat Dec 15 22:36:30 2018 -0500
473
474     Added canvas_node pointer walk function, struct definitions to mirror canvas,
          canvas_node, point, curve in codegen, cleaned up some stuff in the SVG C code
475
476 commit c0f1cd297c863557dfc2e4310e51c8837d5821ca
477 Author: Daniel Mesko <dpmesko@gmail.com>
478 Date:   Sat Dec 15 17:38:35 2018 -0500
479
480     Adding/modifying tests for complex type constructor usage and assignment
481
482 commit a7ece0dd6de15449576de4f41cdba3e573df454f
483 Author: Daniel Mesko <dpmesko@gmail.com>
484 Date:   Sat Dec 15 17:35:47 2018 -0500
485
486     cleaning up unnecessary files
487
488 commit f9de2aa23fa4fbb45943cafee8543e873eb0d252
489 Author: Daniel Mesko <dpmesko@gmail.com>
490 Date:   Sat Dec 15 17:35:18 2018 -0500
491
492     [WIP] Trying to fix identifiers not being valid arguments to type constructors
493
494 commit d9fc1599f3de66d825a81cf8a6d59f3b18a696c7
495 Merge: 5dfaaea 35289cd
496 Author: Daniel Mesko <dpmesko@gmail.com>
497 Date:   Sat Dec 15 15:14:12 2018 -0500
498
499     getting new symbol table, Field access from semant
500
501     Merge branch 'master' into codegen-maddy
502
503 commit 35289cd2154a4529cab9c5a1491a76cff94f1c3c
504 Merge: fe7a81b ec7df3a
505 Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
506 Date:   Sat Dec 15 15:12:52 2018 -0500
507
508     Merge pull request #18 from dpmesko/semant-dan
509
```

```
510     Reverted to old symbol table format, created member_map_of_type funct
511
512 commit 5dfaaea8b3ea67d6a4005a0f73d876b5f9713fa0
513 Author: mtipp <mrt2148@barnard.edu>
514 Date:   Sat Dec 15 14:59:18 2018 -0500
515
516     spacing changes.
517
518 commit 14e1ba41a0aa681c510f181865675c1dd7d4810c
519 Merge: 6db26a6 fe7a81b
520 Author: mtipp <mrt2148@barnard.edu>
521 Date:   Fri Dec 14 21:54:30 2018 -0500
522
523     Merge branch 'master' of https://github.com/dpmesko/ssol into codegen-maddy
524
525 commit fe7a81b84d6b3a5e9f3a1b1f7c453aef3f4b61f2
526 Author: mtipp <mrt2148@barnard.edu>
527 Date:   Fri Dec 14 21:50:00 2018 -0500
528
529     Canvas test passes with int expressions.
530
531 commit ec7df3add0cb1d173419c79b8fd71c3b7303574b
532 Author: Daniel Mesko <dpmesko@gmail.com>
533 Date:   Fri Dec 14 12:16:58 2018 -0500
534
535     Reverted to old symbol table format, created member_map_of_type function,
          implemented Field expr check, removed some unused variables, commented out
          code
536
537 commit 6db26a607bc21826ebd6b404f47240809b3e7f94
538 Author: mtipp <mrt2148@barnard.edu>
539 Date:   Thu Dec 13 18:57:36 2018 -0500
540
541     Canvas constructor compiles.
542
543 commit 75c0cc6122c7748be9841271ec983126fb2e6360
544 Merge: cf00a91 dca0005
545 Author: mtipp <mrt2148@barnard.edu>
546 Date:   Thu Dec 13 16:09:12 2018 -0500
547
548     Merge branch 'master' of https://github.com/dpmesko/ssol into codegen-maddy. Doesn
          't compile.
549
550 commit cf00a91a59a53489381a02581e7f0681d0dd9eb0
551 Author: mtipp <mrt2148@barnard.edu>
552 Date:   Thu Dec 13 16:06:48 2018 -0500
553
554     Start work on canvas constructor.
555
556 commit dca00055fa9d0e78ed7ed462f2a2e8252e060bd2
557 Merge: 1937a11 96c0c73
558 Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
559 Date:   Wed Dec 12 19:02:13 2018 -0500
560
561     Merge pull request #17 from dpmesko/semant-dan
562
563     Finished array assignment, array literals, array access
564
565 commit 96c0c7336b8818a56488a03434d56c5a99906f7b
566 Author: Daniel Mesko <dpmesko@gmail.com>
567 Date:   Wed Dec 12 18:57:58 2018 -0500
568
569     Fixed check_assign for arrays to also check sizes. Added an appropriate testcase
          to array.ssol
570
571 commit d54b35dbdb88e3c8c8b719427fc5d81d0076543c
572 Merge: 5165c8f 19e555c
573 Author: Daniel Mesko <dpmesko@gmail.com>
574 Date:   Wed Dec 12 18:48:50 2018 -0500
575
576     Merging in recent array semant/codegen progress, fixing conflicts
577
```

```
commit 19e555c0841e57dca9696bcf3edd1065e9ba160b
Author: Jeevan <jeevanfarias@gmail.com>
Date:   Wed Dec 12 18:40:04 2018 -0500

    debugged array access, vdeclassign, arraylit

commit 5165c8f835f844db3657ad94598ba653b9b4f81a
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Wed Dec 12 18:38:59 2018 -0500

    Commenting out to get it to compile, to merge in recent Array progress

commit 52f8c90f430f949efd62637bd0fa3a98dc9024f6
Merge: b420194 1937a11
Author: mtipp <mrt2148@barnard.edu>
Date:   Wed Dec 12 17:58:54 2018 -0500

    Merge branch 'master' of https://github.com/dpmesko/ssol into codegen-maddy

commit b420194c250b799680a6a6386446a431e7b93bce
Author: mtipp <mrt2148@barnard.edu>
Date:   Wed Dec 12 17:56:47 2018 -0500

    use named_struct_type in canvas ltype def.

commit b716e3afc4d0e2c5d78bbd1a265f8d8bcc964a4d
Author: mtipp <mrt2148@barnard.edu>
Date:   Wed Dec 12 17:06:49 2018 -0500

    Start defining canvasnode_t in codegen.

commit 4bb43cfcd364efa31205ff19505b90ff72c16e29
Author: Jeevan <jeevanfarias@gmail.com>
Date:   Wed Dec 12 16:46:35 2018 -0500

    added codegen ArrayLit

commit 820dc08378373c141d81d4eb5c0a9b65260be13d
Merge: 80845b6 a5871b4
Author: Jeevan <jeevanfarias@gmail.com>
Date:   Wed Dec 12 16:20:00 2018 -0500

    Merge branch 'semant-dan' of https://github.com/dpmesko/ssol into codegen-expr

commit a5871b4ce260510f5610c6c718a94fc0e6d375e6
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Wed Dec 12 16:19:40 2018 -0500

    Fixed arraylit checking

commit 80845b670b079d6fa7c49c8415488781f997258b
Merge: 60fdc57 1937a11
Author: Jeevan <jeevanfarias@gmail.com>
Date:   Wed Dec 12 16:16:07 2018 -0500

    merging array lit changes from master

commit 60fdc57b21566a308b6f2aac48fd6c19c7ff4e64
Author: Jeevan <jeevanfarias@gmail.com>
Date:   Wed Dec 12 16:14:10 2018 -0500

    started codegen expr arraylit

commit 1937a110a359a268a9d240506ea905031d210b83
Merge: 6019a00 05ddebf
Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
Date:   Wed Dec 12 15:59:02 2018 -0500

    Merge pull request #16 from dpmesko/semant-dan

    arrays in semant, decls and constructors in codegen
```

```
commit 05ddebfe1eb8b6b1bfaadbc30eb45de524ea69e0
Merge: 17a6ad8 6019a00
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Wed Dec 12 15:51:19 2018 -0500

    Merging master, fixing conflicts, deleting backups/assembly files

commit 17a6ad800d1d807c86947e29599997d155756720
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Wed Dec 12 15:43:26 2018 -0500

    Commented out broken symbol map code to get it to compile

commit 6019a000bae218e478a452e83b1db452ddc19f3f
Author: jvntf <jeevanfarias@gmail.com>
Date:   Wed Dec 12 15:30:26 2018 -0500

    Semant adecl (#15)

    * adecl codegen written

    * added failed tests.

    * fixed semantic checking of adecl

    * Added local mapping to pt and curve constructors. does not make draw.o or svg.o
        ?

    * Point assignment passes test pt-assign.ssol

    * Curve only passes test with point constductors.

    * fixed ADecl and ADecl ast type

commit a5551e812d53c096b20f1e661f01c8818d5da741
Author: mtipp <mrt2148@barnard.edu>
Date:   Tue Dec 11 18:34:14 2018 -0500

    Curve only passes test with point constductors.

commit eedf70454671cca47acf7c905374080c44fbfaac
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Tue Dec 11 18:11:51 2018 -0500

    [WIP] updating Field pattern to work with new symbol table, fixed Mod pattern

commit 22310bcc4a3bbf8771dcfa5110157f28e9d2eaee
Author: mtipp <mrt2148@barnard.edu>
Date:   Tue Dec 11 18:03:35 2018 -0500

    Point assignment passes test pt-assign.ssol

commit 43243b64945186f05cb5ddb85604e4b894af79b9
Author: mtipp <mrt2148@barnard.edu>
Date:   Tue Dec 11 17:46:18 2018 -0500

    Added local mapping to pt and curve constructors. does not make draw.o or svg.o ?

commit 89e93c5b84034e74e663ef796d69d817211b476b
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Tue Dec 11 16:43:39 2018 -0500

    Added semantic check for Mod

commit a82209cd2bdf7a517d65d7888316a4164bf0dd78
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Tue Dec 11 16:33:26 2018 -0500

    Updated creation of initial symbol table to have (type, option StringMap) as
        values, with member variable maps for complex types

commit f3b368c49b07820ffb796f2bccd886c7c02ea8a5
```

```
720 Merge: 73ee207 1cf9203
721 Author: mtipp <mrt2148@barnard.edu>
722 Date:   Tue Dec 11 15:40:14 2018 -0500
723
724     Merge branch 'master' of https://github.com/dpmesko/ssol into codegen-maddy
725
726 commit 1cf9203afcd8aa3d7c612a6c79cb42d4d9ee6a35
727 Merge: 57f3eb6 b547b5c
728 Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
729 Date:   Tue Dec 11 15:34:46 2018 -0500
730
731     Merge pull request #14 from dpmesko/tuesday-merge
732
733     Semantic checking and codegen for expressions and statements
734
735 commit 73ee207926b045d354c0ccb965f4a4023d787e01
736 Author: mtipp <mrt2148@barnard.edu>
737 Date:   Tue Dec 11 15:34:46 2018 -0500
738
739     added failed tests.
740
741 commit f92881552adb5f0e2b198f54575ae8fb443ac9e7
742 Author: Daniel Mesko <dpmesko@gmail.com>
743 Date:   Tue Dec 11 15:29:46 2018 -0500
744
745     messing with initial StringMap
746
747 commit 2fc3dc5a6e6969c7e9e258eace9fdbe6f6c1f4af
748 Author: Daniel Mesko <dpmesko@gmail.com>
749 Date:   Sun Dec 9 17:48:39 2018 -0500
750
751     Uncommented test script target in Makefile
752
753 commit 1c784d2562564f1de9b05592f598fb5950f3803b
754 Author: Daniel Mesko <dpmesko@gmail.com>
755 Date:   Sun Dec 9 17:48:07 2018 -0500
756
757     Implemented pipe, pipend, arraylit patterns; began converting symbol map to name
        -> (ty, option StringMap), passing as arg to expr, member map access function
758
759 commit b547b5c7d18fc14882c09338129096a547fd406e
760 Author: Daniel Mesko <dpmesko@gmail.com>
761 Date:   Thu Dec 6 22:48:32 2018 -0500
762
763     Removed lambda map build nonsense, got it to compile
764
765 commit 0f7a0a758d201716cc29021c834ecbf3f76a7c77
766 Author: Daniel Mesko <dpmesko@gmail.com>
767 Date:   Thu Dec 6 17:14:26 2018 -0500
768
769     added new param to expr calls
770
771 commit 0e2f730c494902304416a36d0564a2b7c2c99672
772 Merge: e096683 bf0976b
773 Author: Daniel Mesko <dpmesko@gmail.com>
774 Date:   Thu Dec 6 17:04:38 2018 -0500
775
776     Last merge to bring me up to date
777
778 commit e096683ebdaa84991531ebd4bddf8eecc8c68112
779 Merge: 5312b24 49e3c93
780 Author: Daniel Mesko <dpmesko@gmail.com>
781 Date:   Thu Dec 6 17:02:06 2018 -0500
782
783     Merging other branches before I do any more duplicate work...
784
785 commit 5312b24ffc50c8b7356242fe198869824cab9e22
786 Merge: 6d8baf3 6843fb6
787 Author: Daniel Mesko <dpmesko@gmail.com>
788 Date:   Thu Dec 6 16:42:01 2018 -0500
789
790     merged in semant-jf
```

```
791
792 commit 6d8baf3fad6256a64673a5a47baafe5828a53728
793 Author: Daniel Mesko <dpmesko@gmail.com>
794 Date:   Thu Dec 6 16:30:53 2018 -0500
795
796     [WIP] modifying symbol table to store (typ, Stringmap) pairs for members of
          complex types; passing current symbol map to expr
797
798 commit 49e3c93306c32bb3645cd784d8aef459296f76db
799 Author: Jeevan <jeevanfarias@gmail.com>
800 Date:   Wed Dec 5 21:48:36 2018 -0500
801
802     var decl and assignment working
803
804 commit d94eb8c1ee7f89faff42a17d6575ca8820511518
805 Author: Jeevan <jeevanfarias@gmail.com>
806 Date:   Wed Dec 5 20:47:00 2018 -0500
807
808     modified semant and codegen to allow for var decl. works but value is wrong
809
810 commit fb548a50fc02a81646d2f797fec97858a574dde6
811 Author: Jeevan <jeevanfarias@gmail.com>
812 Date:   Wed Dec 5 19:48:20 2018 -0500
813
814     added localvar map to codegen->expr, need to make semant mods to make work
815
816 commit 8868393260051249d07ae692f7e5f02f914861bd
817 Author: Jeevan <jeevanfarias@gmail.com>
818 Date:   Wed Dec 5 18:18:24 2018 -0500
819
820     commented out ADecl
821
822 commit e144d08c69d8eee28f44d9ff66d9787ee398436e
823 Author: Jeevan <jeevanfarias@gmail.com>
824 Date:   Tue Dec 4 21:22:11 2018 -0500
825
826     added VDeclAssign to codegen. NEED TO CHANGE expr to take local string map
827
828 commit edd053dc9851db4187fd58699c6612882cd821ef
829 Author: Jeevan <jeevanfarias@gmail.com>
830 Date:   Tue Dec 4 12:04:23 2018 -0500
831
832     added a local var map to codegen.stmt and added SVDecl pattern
833
834 commit 6843fb6b5a1a4e1f01d8a7b0c6e0a5bc79be6bfa
835 Author: Jeevan <jeevanfarias@gmail.com>
836 Date:   Wed Dec 5 19:36:02 2018 -0500
837
838     fixed semant:: stmt -> SBlock
839
840 commit bf0976be959d559434f070d9430e176fb4ddf4d5
841 Author: mtipp <mrt2148@barnard.edu>
842 Date:   Tue Dec 4 15:41:12 2018 -0500
843
844     SConstructor for point compiles
845
846 commit 6f400d491640997c9323ffa7ab38f3a1d3d3a533
847 Author: Jeevan <jeevanfarias@gmail.com>
848 Date:   Mon Dec 3 22:54:29 2018 -0500
849
850     matched pattern on [], need to verify that this works
851
852 commit 17b8b9f8f7f5777886a0d08fc4dd76c4b6ad4a89
853 Author: Jeevan <jeevanfarias@gmail.com>
854 Date:   Mon Dec 3 16:08:53 2018 -0500
855
856     fixed partial function appl
857
858 commit d95d8edef181fc1473213ca757e099182f47c785
859 Author: Jeevan <jeevanfarias@gmail.com>
860 Date:   Mon Dec 3 16:07:29 2018 -0500
861
```

```
862       added block_locals to stmt
863
864   commit c7c44c381438d56cfddbde720f063184eaca127a
865   Author: Daniel Mesko <dpmesko@gmail.com>
866   Date:   Mon Dec 3 00:13:17 2018 -0500
867
868       complex type constructor calls in semant, pretty printing in ast, sast
869
870   commit cb898bc2efd2ad506d86bd8712d064f2f3c7494d
871   Author: Jeevan <jeevanfarias@gmail.com>
872   Date:   Sat Dec 1 17:34:48 2018 -0500
873
874       added VDdecl, VDeclAssign, ADecl to semant
875
876   commit 57f3eb6ada797b2d5597bebb54b56b34f3b5d734
877   Author: jvntf <jeevanfarias@gmail.com>
878   Date:   Wed Nov 28 13:43:36 2018 -0500
879
880       changes from hello world (#10)
881
882       * wrote readme, ready for submission
883
884       * removed locals and vdecls_list from fdecl
885
886       * removed refs to locals from ast,sast, semant
887
888       * edit removal of locals
889
890       * fixed type problems in parser, still need to handle alloc in codegen
891
892       * added new zip file
893
894       * removed zip file from repo
895
896   commit 7e38cbce012137b53fece9a47ccaf06e429e16ba
897   Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
898   Date:   Mon Nov 19 01:50:14 2018 -0500
899
900       Update README.md
901
902   commit 185362785e7d6aecf716c9d6d7063e95c0cb2134
903   Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
904   Date:   Mon Nov 19 01:49:38 2018 -0500
905
906       Update README.md
907
908   commit b3bdc72d5cf3439aa0002ba1f9ed31f6465f7557
909   Author: Jeevan <jeevanfarias@gmail.com>
910   Date:   Mon Nov 19 01:36:16 2018 -0500
911
912       added new zip file
913
914   commit 92ad5e0cc5014a0508027097e2633ae8911842da
915   Author: Jeevan <jeevanfarias@gmail.com>
916   Date:   Mon Nov 19 01:23:20 2018 -0500
917
918       fixed type problems in parser, still need to handle alloc in codegen
919
920   commit 77c44a312a3be67aacc6fe49170125c69b1abf74
921   Author: Jeevan <jeevanfarias@gmail.com>
922   Date:   Mon Nov 19 01:12:15 2018 -0500
923
924       edit removal of locals
925
926   commit 9324e1fa99ae558cced68a1a51836de30aa3e4e2
927   Author: Jeevan <jeevanfarias@gmail.com>
928   Date:   Mon Nov 19 00:50:01 2018 -0500
929
930       removed refs to locals from ast,sast, semant
931
932   commit 382c20fbe7c798597cc7d914318f2ae0344a8f24
933   Author: Jeevan <jeevanfarias@gmail.com>
```

```
Date:   Mon Nov 19 00:49:30 2018 -0500

    removed locals and vdecls_list from fdecl

commit 589b0dffa8f45d91037c257681011a4dd2e7db4b
Author: Jeevan <jeevanfarias@gmail.com>
Date:   Wed Nov 14 18:30:29 2018 -0500

    wrote readme, ready for submission

commit db59506350de0b861bb70643feab14c513afa481
Merge: 338d8a4 5c630fc
Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
Date:   Tue Nov 13 21:23:34 2018 -0500

    Merge pull request #9 from dpmesko/draw-link

    added SVG library linking, hello-world test draw

commit 5c630fc0cfbd5f0a2b31a4cc5284a88edef9e897
Merge: d6eb83f 338d8a4
Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
Date:   Tue Nov 13 21:23:19 2018 -0500

    Merge branch 'master' into draw-link

commit 338d8a47f708bf177c2e9526146dbeed33561f96
Merge: 7b656c4 39931bb
Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
Date:   Tue Nov 13 21:21:46 2018 -0500

    Merge pull request #8 from dpmesko/semant-dan

    Added semantic checking for expressions, codegen

commit d6eb83fd8261c57f72e4becfe18e25bfc097ed24
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Tue Nov 13 21:19:41 2018 -0500

    Modified svg.c to format string properly, added hello world test draw and LLVM
        compilation script

commit cb13cac3b5ca7a3979f4d173e0c1f125991e6325
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Tue Nov 13 21:18:45 2018 -0500

    Added filename to draw() function, changed built-in function args to be lists, to
        support multiple args

commit 5e3539394e680a89a5e3862e34ff5b93c7e480d1
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Tue Nov 13 21:18:08 2018 -0500

    Added filename arg to draw function

commit 365e2f29a92e7e7c1ce07fe97961534527849d45
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Tue Nov 13 21:17:46 2018 -0500

    Added filename argument to draw() in codegen

commit d73dc759f76e264a646aa52ea5ae36785899f47f
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Tue Nov 13 21:17:25 2018 -0500

    Removed clean from all target

commit d599802eaa1829cf64e9e1fd1b1648c2a001ac22
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Tue Nov 13 20:57:09 2018 -0500

    Added clean to all target
```

```
commit 39931bb6a437b95dd3c498c654b9de94daab9394
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Tue Nov 13 20:33:49 2018 -0500

    Added clean dependency to all target, to remove old object files when remaking

commit 3b4cfb860d3e4e20bb78434ca603f5e41d15732b
Author: Jeevan <jeevanfarias@gmail.com>
Date:   Tue Nov 13 12:15:10 2018 -0500

    hello world

commit bd8e9efa44e5cf2dfe2ca069db70a2ffbb8c7f1e
Author: Jeevan <jeevanfarias@gmail.com>
Date:   Tue Nov 13 11:42:17 2018 -0500

    wrote comp script, baby test program

commit 3ba54a9d339e37fd3d5ccf89f6b031adc46d9ae7
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Mon Nov 12 21:13:33 2018 -0500

    Moved draw C files

commit f8fdd7a0dab1556b811edbb6363087303270c57b
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Mon Nov 12 21:13:14 2018 -0500

    Added SVG drawing C code, started attempting to link it with the compiler

commit 821bffaa5839e5a45a588fd47529be46da5746be
Merge: 2f8a8f4 13d80fb
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Mon Nov 12 20:57:36 2018 -0500

    I WANT DA NEW STUFF

commit 2f8a8f4aae3e84e33a5af3894a9ae2cf1a1aafd8
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Mon Nov 12 20:56:52 2018 -0500

    Added built-in draw() function, changed testall.sh to use new executable name

commit dc6f838bd1520af4cd5b6490730a176c0a819a8a
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Mon Nov 12 19:57:11 2018 -0500

    Changes to ast, sast, semant, parser, to add Complex types constructors and member
        access

commit bda9376295154e6ec0c8c3c4caf1e9f7f288ca7c
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Mon Nov 12 19:34:01 2018 -0500

    Added expr rule for complex type constructors

commit 13d80fb8c89c05735c73592f4c7bd0ac00d4645f
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Sun Nov 11 17:15:04 2018 -0500

    Created C functions for writing SVG files, to be linked at LLVM codegen stage, for
        use with draw() under the hood

commit 4bf55301510952017737254dae4760b9e73df510
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Sun Nov 11 16:49:14 2018 -0500

    Fixed spacing

commit 522aa609c8364e752f28ce51337d90890d31097b
Merge: 8eb4e2b e736316
```

```
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Sun Nov 11 16:17:38 2018 -0500

    fixed merge conflicts with Maddy's branch

commit e736316a88fbb563e0db6a4d5b456f414fc601a7
Merge: 21900f8 4882230
Author: mtipp <mrt2148@barnard.edu>
Date:   Sun Nov 11 16:13:54 2018 -0500

    Merge branch 'semant-maddy' of https://github.com/dpmesko/ssol into semant-maddy

commit 21900f8567b47558a6011585cd6c4b6b87594c89
Author: mtipp <mrt2148@barnard.edu>
Date:   Sun Nov 11 15:07:42 2018 -0500

    add mod sbinop.

commit 8eb4e2b93631562e3a5aaaf32afbf18dbb0b2926
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Sun Nov 11 16:12:00 2018 -0500

    [WIP] Implementing ArrayAssign, ArrayAccess expressions in semantic checker

commit 4882230edf2b681ad0ca3262eca1270428557dd2
Author: mtipp <mrt2148@barnard.edu>
Date:   Sun Nov 11 15:07:42 2018 -0500

    add mod sbinop.

commit eaeda3cf7d82606ca5881be63f671291719303fa
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Sun Nov 11 14:52:39 2018 -0500

    Fixed ArrayLit logic, check_assign for arrays

commit 273f54489b3a54a3d5f194ff7ab7aa0450fbd834
Author: mtipp <mrt2148@barnard.edu>
Date:   Sun Nov 11 14:30:08 2018 -0500

    start adding sprint to codegen.

commit 70dfeaad8a84366b9de16b210ab3823b5912537d
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Sun Nov 11 14:00:19 2018 -0500

    added ArrayLit logic

commit 99ec9be514bb81c015e05e68b642668fdbda3a1c
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Sun Oct 28 19:46:39 2018 -0400

    Added TODOs

commit 2e9d6b58e5808a795318f5ace097461cdcd12ee9
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Sun Oct 28 19:26:28 2018 -0400

    Added combined declaration/assignment, array declaration, and their sast
        counterparts

commit 7b656c4a24b7ff19b08b57c2a6d4f344ce26c037
Merge: cd6324f 76fb8ad
Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
Date:   Fri Oct 26 16:27:10 2018 -0400

    Merge pull request #7 from dpmesko/new-microc

    Merge SSOL scanner/parser/ast with newest MicroC source

commit 76fb8ad7ebc8e776e2744a00a3c7b3d5dc1b2bfc
Author: Daniel Mesko <dpmesko@gmail.com>
```

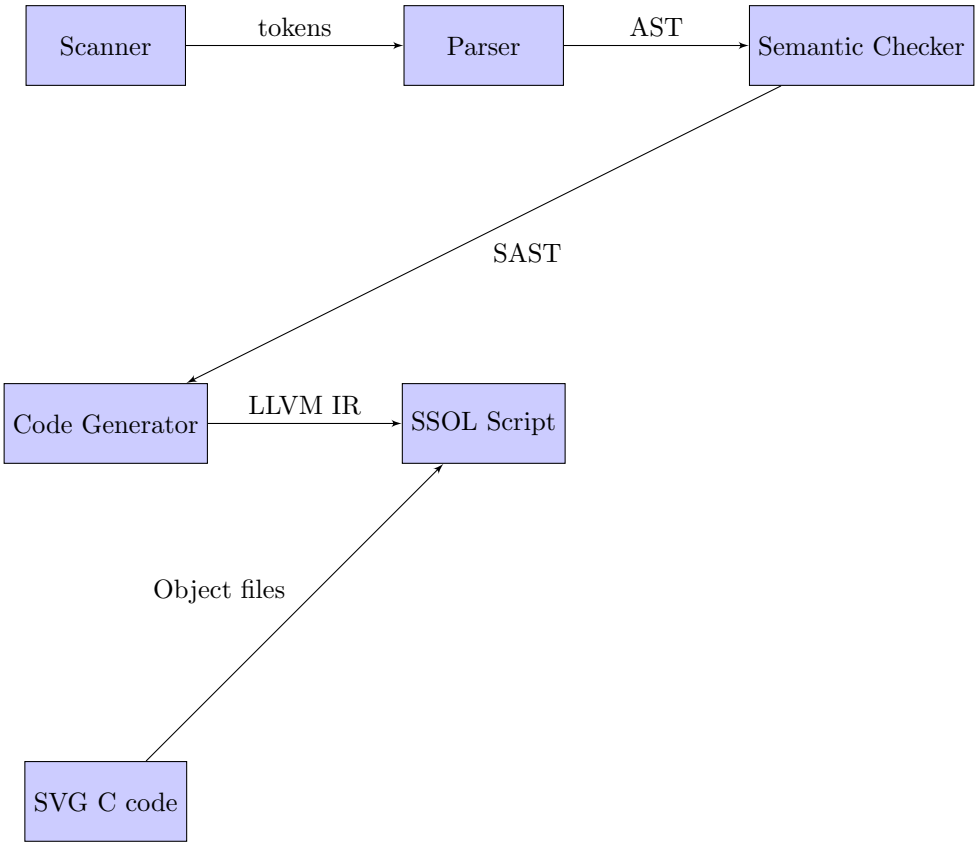
```
Date:   Fri Oct 26 16:09:53 2018 -0400

    Merged SSOL scanner/parser/ast with newest MicroC source

commit cd6324f65e8ec5703fc30c696683c5d3028ba0a7
Merge: 7b3230d b088278
Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
Date:   Wed Oct 17 20:42:24 2018 -0400

    Merge pull request #6 from dpmesko/parser-dan

    scanner, parser, ast - preliminary versions

commit b088278e6995b1e837b689b090c2ce3e67ca2943
Author: Jeevan <jeevanfarias@gmail.com>
Date:   Mon Oct 15 19:35:46 2018 -0400

    updated lrm

commit e3a43185d28767905d572d101d4aed3ec0ece6c0
Merge: af8626b e2d409f
Author: Jeevan <jeevanfarias@gmail.com>
Date:   Mon Oct 15 19:34:13 2018 -0400

    Merge branch 'parser-dan' of https://github.com/dpmesko/ssol into parser-dan

commit af8626be6b8c70437ddb0e6f35e32f2653e5d422
Author: Jeevan <jeevanfarias@gmail.com>
Date:   Mon Oct 15 19:15:10 2018 -0400

    added array access

commit e2d409ff1c1711ad36f5dd20546d813c2aa7d677
Author: Jeevan <jeevanfarias@gmail.com>
Date:   Mon Oct 15 19:15:10 2018 -0400

    added array access

commit 9f164636e47a4b03609161befed4d3628da7a5cf
Author: Jeevan <jeevanfarias@gmail.com>
Date:   Mon Oct 15 18:41:50 2018 -0400

    added newest version of lrm

commit 382c57950166fa536ff56a7d7b6524bd9f1d32f2
Author: Jeevan <jeevanfarias@gmail.com>
Date:   Mon Oct 15 18:39:26 2018 -0400

    fixed assoc

commit f1cfca030786e9c6b22263ed771603d1fe3929a4
Author: Jeevan <jeevanfarias@gmail.com>
Date:   Mon Oct 15 18:34:16 2018 -0400

    added object fields to parser and ast

commit 275cc285df4296d083ae14e61e81038371e39f9c
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Mon Oct 15 18:11:45 2018 -0400

    Updated Makefile dependencies, added array literals to ast

commit 15d43e2545954e2937687b3cd30c3becdca9f5d8
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Sun Oct 14 19:45:44 2018 -0400

    [WIP] Partially added arrays, added break and continue, pretty print patterns

commit 113f9dc363ba67b943949e6c1017761402990c02
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Sun Oct 14 18:19:19 2018 -0400
```

```
1217      Added char literals, changed file names in Makefile, renamed stuff....
1218
1219  commit b2823c384eb6ea68c3180090683a227694bcc61b
1220  Author: Daniel Mesko <dpmesko@gmail.com>
1221  Date:   Sat Oct 13 18:42:59 2018 -0400
1222
1223      [WIP] Added (some) operator types to ast
1224
1225  commit a3c47ac1c03c86344ef90b23e474ecbe82fc605c
1226  Author: Daniel Mesko <dpmesko@gmail.com>
1227  Date:   Sat Oct 13 18:42:07 2018 -0400
1228
1229      Added tokens to parser, updated typ rule, added MOD to expr rule
1230
1231  commit cf845b6a65a80c15eb04b136eed8a067b08f3a80
1232  Author: Daniel Mesko <dpmesko@gmail.com>
1233  Date:   Sat Oct 13 18:41:13 2018 -0400
1234
1235      Added missing }, renamed LITERAL token to INT_LITERAL
1236
1237  commit 7b3230de3c511c55ebe184babf7187e30ccd9ee4
1238  Merge: 4321218 5fd81bb
1239  Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
1240  Date:   Fri Oct 12 16:07:07 2018 -0400
1241
1242      Merge pull request #4 from dpmesko/scanner
1243
1244      added (probably) all tokens to scanner
1245
1246  commit 4321218c0be09af0488de631fafee00c5feb7011
1247  Author: Jeevan <jeevanfarias@gmail.com>
1248  Date:   Fri Oct 12 16:03:29 2018 -0400
1249
1250      added lrm pdf
1251
1252  commit 95be44d93414c4a06efff772d5d1555b29f97ad2
1253  Merge: 334b93b 8176066
1254  Author: jvntf <jeevanfarias@gmail.com>
1255  Date:   Fri Oct 12 15:59:31 2018 -0400
1256
1257      Merge pull request #5 from dpmesko/lrm
1258
1259      added lrm latex source
1260
1261  commit 81760663314d887c24df260be36ea1a57125a0ec
1262  Author: Jeevan <jeevanfarias@gmail.com>
1263  Date:   Fri Oct 12 15:59:08 2018 -0400
1264
1265      added lrm latex source
1266
1267  commit 5fd81bb8d8a2444138447c6851c8c94daf03ed0c
1268  Author: Daniel Mesko <dpmesko@gmail.com>
1269  Date:   Fri Oct 12 14:37:29 2018 -0400
1270
1271      Made it so that float literals can omit the integer part, added DOT token
1272
1273  commit c487df90263b87dab885399d6d8540d820edf2e4
1274  Author: Daniel Mesko <dpmesko@gmail.com>
1275  Date:   Wed Oct 10 21:03:15 2018 -0400
1276
1277      Added more elements, single line comments
1278
1279  commit cfd403b78493eab2824d2ea3874140181cd30c35
1280  Author: Daniel Mesko <dpmesko@gmail.com>
1281  Date:   Wed Oct 10 20:39:52 2018 -0400
1282
1283      Began adding token rules to scanner
1284
1285  commit 334b93bc40921a5d9ac24e7a70df3854a06b33eb
1286  Merge: 91c88a3 08b7d22
1287  Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
1288  Date:   Fri Oct 5 19:52:52 2018 -0400
```

```
1289
1290        Merge pull request #3 from dpmesko/proposal
1291
1292        Proposal merge
1293
1294    commit 91c88a348f0ebf7d0cc0ab68f28e52da3b2c922c
1295    Merge: 5431dd5 5116d87
1296    Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
1297    Date:   Fri Oct 5 19:48:44 2018 -0400
1298
1299        Merge pull request #2 from dpmesko/setup
1300
1301        Added MicroC compiler source, renamed microc.ml, now ssol.ml
1302
1303    commit 5116d87d184ba96004a0780bba47abb6c365634d
1304    Author: Daniel Mesko <dpmesko@gmail.com>
1305    Date:   Fri Oct 5 19:45:28 2018 -0400
1306
1307        Added MicroC compiler source, renamed microc.ml, now ssol.ml
1308
1309    commit 5431dd502d3f6747db76d3a106b34d8ea66b6a49
1310    Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
1311    Date:   Wed Oct 3 01:02:38 2018 -0400
1312
1313        Update README.md
1314
1315    commit 08b7d225e0b0519d1c06f4c8241bf7a934e28408
1316    Author: Jeevan <jeevanfarias@gmail.com>
1317    Date:   Wed Sep 19 19:45:21 2018 -0400
1318
1319        capitalized
1320
1321    commit 32b89a241676189bebafec381f2be8ac49a6e474
1322    Author: Jeevan <jeevanfarias@gmail.com>
1323    Date:   Wed Sep 19 19:32:25 2018 -0400
1324
1325        yadl sample program
1326
1327    commit d371c7fc267c5eba7a9d7d35b8b55ef00f2dec83
1328    Merge: 163c583 4874d18
1329    Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
1330    Date:   Sun Sep 16 18:02:36 2018 -0400
1331
1332        Merge pull request #1 from dpmesko/proposal
1333
1334        Sample Program for Proposal
1335
1336    commit 4874d18f8b22db6ceb1716d467d4a1371c8fb8fc
1337    Author: Jeevan <jeevanfarias@gmail.com>
1338    Date:   Sun Sep 16 17:37:20 2018 -0400
1339
1340        added for loop
1341
1342    commit 163c583e00ea9fcb3520bb55ab23fa4de1415a4c
1343    Author: Jeevan <jeevanfarias@gmail.com>
1344    Date:   Sun Sep 16 17:29:33 2018 -0400
1345
1346        started sample program
1347
1348    commit cd934b1874bc4ccb54710342bcaf1bc1b9ddb36d
1349    Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
1350    Date:   Sun Sep 16 17:14:21 2018 -0400
1351
1352        Initial commit
```

Chapter 4

Architectural Design



4.1 Interfaces

The block diagram above shows the various components of our compiler and the interfaces between them. An input SSOL program is fed into the scanner, tokenized, and passed to the parser, which generates the abstract syntax tree. The AST is passed to the semantic checker, which generates a semantically-checked AST (SAST - an AST with types associated with each expression). The SAST is passed to the code generator, which generates the LLVM IR of the source program. At this point, the supplied `./ssol` script compiles the LLVM IR into a `.s` native assembly code file, and links it with the C code object files (compiled when `make` is run. Finally, an executable file is generated and run by the script, before being removed, along with the generated `.ll` and `.s` files.

4.2 Error Checking

If a source program contains invalid tokens, the scanner rejects it. If the source program contains syntactically invalid statements or expressions, the parser rejects it. If the source program contains semantically invalid statements or expressions, the semantic checker rejects it.

4.3 Authorship

The scanner and parser were implemented collectively by all three team members. Semantic checking for expressions was written by Daniel, semantic checking for statements was written by Jeevan, and code generation for expressions was written by Maddy. Code generation for statements was written by Jeevan and Daniel. The logic for the `pipend` operator in code generation was written by all three team members. The SSOL top-level script was written by Jeevan. The SVG C code was written by Daniel.

Chapter 5

Test Plan

Each new feature was tested with passing and failing tests before being merged into the master branch of the repository. These new features, such as built-in functions, constructors, or complex-types, were tested both individually and with the rest of the test suite to ensure they didn't break any other parts of the project.

The convention we followed was that the passing tests would either produce output to match a .out file, or produce no output. The failing tests were compared with their corresponding .err files to ensure expected compiler error messages. Madeleine, the designated Test Architect, wrote the majority of the tests for new features, but all team members contributed to tests for features that they helped develop.

5.1 SSOL to LLVM

Here are examples of the tests used to test the `Curve` constructor. We tested the `Curve` assignment feature separately, as its constructor was defined separately from assignment in codegen.ml.

test-curveconstruct.ssol:

```
int main() {
  Curve c = Curve(Point(1.0, 1.0), Point(1.0, 1.0Point(1.0, 1.0), Point(1.0, 1.0));
  return 0;
}
```

The LLVM output from test-curveconstruct.ssol:

```
; ModuleID = 'SSOL'
source_filename = "SSOL"

%canvasnode = type { %canvasnode*, { { double, double }, { double, double }, { double, double }, { double,

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%g\0A\00"

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

declare i32 @printbig(i32)

declare i32 @draw({ double, double, %canvasnode* }, i8*)

declare { double, double } @Point(double, double)

declare { { double, double }, { double, double }, { double, double }, { double, double } } @Curve({ double
```

```
declare { double, double, %canvasnode* } @Canvas(double, double)

define i32 @main() {
entry:
  %c = alloca { { double, double }, { double, double }, { double, double }, { double, double } }
  %Point = call { double, double } @Point(double 1.000000e+00, double 1.000000e+00)
  %Point1 = call { double, double } @Point(double 1.000000e+00, double 1.000000e+00)
  %Point2 = call { double, double } @Point(double 1.000000e+00, double 1.000000e+00)
  %Point3 = call { double, double } @Point(double 1.000000e+00, double 1.000000e+00)
  %Curve = call { { double, double }, { double, double }, { double, double }, { double, double } } @Curve(
  store { { double, double }, { double, double }, { double, double }, { double, double } } %Curve, { { dou
  ret i32 0
}
```

We then ensured that the `Curve` constructor would fail semantic checking for number of arguments and type of arguments in fail-curveconstruct-numargs.ssol and fail-curveconstruct-typeargs.ssol, respectively. We ensured expected failure by comparing the outputs in their `.err` files.

fail-curveconstruct-numargs.ssol:

```
int main(){
  Curve(30, 30);
  return 0;
}
```

fail-curveconstruct-numargs.err:

```
Fatal error: exception Failure("expecting 4 arguments in Curve(30, 30)")
```

The LLVM output from fail-curveconstruct-numargs.ssol:

```
Fatal error: exception Failure("expecting 4 arguments in Curve(30, 30)")
```

fail-curveconstruct-typeargs.ssol:

```
int main(){
  Curve(Point(1.0, 1.0), Point(2.0, 2.0), Point(3.03.0), 4.0);
  return 0;
}
```

fail-curveconstruct-numargs.err:

```
Fatal error: exception Failure("illegal argument found float expected Point in 4.0")
```

The LLVM output from fail-curveconstruct-numargs.ssol:

```
Fatal error: exception Failure("illegal argument found float expected Point in 4.0")
```

5.2 Test Suite

Our test suite is built off of the MicroC test suite. The test suite can be run with the testall.sh script, and is run with the make command.

testall./sh:

```
#!/bin/sh

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

```
#  Compile and check the error of each expected-to-fail test

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

# Path to the LLVM compiler
LLC="llc"

# Path to the C compiler
CC="cc"

# Path to the SSOL compiler.  Usually "./ssol.native"
# Try "_build/ssol.native" if ocamlbuild was unable to create a symbolic link.
SSOL="./ssol.native"
#SSOL="_build/ssol.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] [.ssol 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/.ssol//’‘
    reffile=‘echo $1 | sed ’s/.ssol$//’‘
    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 ${basename}.out" &&
    Run "$SSOL" "$1" ">" "${basename}.ll" &&
    Run "$LLC" "-relocation-model=pic" "${basename}.ll" ">" "${basename}.s" &&
    Run "$CC" "-o" "${basename}.exe" "${basename}.s" "printbig.o" "draw.o" "svg.o" &&
    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/.ssol//’‘
    reffile=‘echo $1 | sed ’s/.ssol$//’‘
```

```
    basedir="‘echo $1 | sed ’s/\/[^\/]*$//’‘/."

    echo -n "$basename..."

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

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&
    RunFail "$SSOL" "<" $1 "2>" "${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 [ ! -f printbig.o ]
then
    echo "Could not find printbig.o"
    echo "Try \"make printbig.o\""
    exit 1
fi

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

```
    files=$@
else
    files="tests/test-*.ssol tests/fail-*.ssol"
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
```

Chapter 6

Lessons Learned

6.1 Daniel

Throughout this project I learned how important careful version control is to development productivity. Initially, we spent a lot of time fixing merge conflicts, rebasing branches, and often doing duplicate work. We decided ultimately to follow the model of branching off of the master branch and making pull requests to merge back in, which were reviewed by all other members. We also made sure to delegate chunks of work so as to prevent overlapping work, and clearly communicate with one another when one team member's part relied on that of another. We also agreed to leave our pull requests open until all related chunks were completed, and then together, in-person, we reviewed each of them before merging. We then all rebranched off of master to make sure our copies were synchronized. This worked very well.

I also learned a great deal about functional programming and how valuable it is (no more seg faults! just type errors ad nauseam), as well as compiler infrastructure and how source code is translated into CPU instructions (very useful for understanding optimization in code writing, even at a high level).

I recommend that future PLT students put in a lot of individual ground work before working on the project (like reading the Dragon book, learning OCaml syntax, etc). This will make you a better team member and make the compiler development process more about cool language/structural decisions and code writing and not about fussy syntax errors. I also recommend over preparing your team with regards to version control and communication channels. Nothing slows down a group project like avoidable logistical problems.

6.2 Maddy

This project was my introduction to functional programming and compiler architecture. To future PLT students, I recommend taking the first homework assignment seriously in order to avoid playing OCaml catch up later in the semester. Looking through the MicroC source code and comparing it to the lecture notes was an effective way to get my bearings on the expected product and compiler architecture overall.

When dividing up the group work we used separate git branches and pull requests to maintain an organized repository. This strategy worked well for us because we could see who made what changes, which made debugging much easier. In the beginning, though, when we were just working on the parser and scanner, we did a lot of partner programming on just one group member's computer and talked out all the decisions. This way, we all had the same base knowledge of the project before we all went off to work separately on semantic checking and code generation.

We spent a lot of time going through the LLVM documentation to no avail when we began working on code generation. It would have been in our best interests to go through those docs with our TA earlier on in the project, which would have made us more efficient when adding new features.

6.3 Jeevan

Aside from the task of producing a compiler itself, navigating the balance of individual and group work is the biggest challenge of this project. It took us a few sessions to develop a rhythm that made us productive but kept everyone in the loop and learning about the develoment process. In my opinion, it was left effective to sit together with one person at the keyboard. This setup was what we defaulted to at the beginning, but I found that it gave me a much shallower understanding of the compiler architecture. I learned the most when I implemented small sections of the compiler throughout the pipeline, as in, building variable declaration logic in both semantic checking and code generation. Of course it is often helpful to talk things out together, but I found this most useful as needed and for very specific problems.

One of the biggest learning moments for me was the understanding of the difference between compile time and runtime while writing the code generation stage of the compiler. The difference between instructions and actual values is a crucial detail for a full understanding of the project.

Chapter 7

Appendix

Below is the source code for each module of our compiler, as well as the test cases we added to the MicroC test suite to test the features we added. The original MicroC tests still remain within the source code repository, and are compiled, run, and verified by the test script, but have been excluded from this appendix.

7.1 Compiler Core

7.1.1 Scanner

```
1  (* Ocamllex scanner for SSOL *)
2  (*      Authors: Jeevan Farias, Madeleine Tipp, Daniel Mesko *)
3
4  { open Parserssol }
5
6  let digit = ['0' - '9']
7  let digits = digit+
8
9  rule token = parse
10   [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
11 | "/*"     { comment lexbuf }           (* Comments *)
12 | "//"     { single lexbuf }            (* Single line comments *)
13 | '('      { LPAREN }
14 | ')'      { RPAREN }
15 | '{'      { LBRACE }
16 | '}'      { RBRACE }
17 | '['      { LBRACK }
18 | ']'      { RBRACK }
19 | ';'      { SEMI }
20 | ','      { COMMA }
21 | '.'      { DOT }
22 | '+'      { PLUS }
23 | '-'      { MINUS }
24 | '*'      { TIMES }
25 | '/'      { DIVIDE }
26 | '%'      { MOD }
27 | '='      { ASSIGN }
28 | "=="     { EQ }
29 | "!="     { NEQ }
30 | '<'      { LT }
31 | "<="     { LEQ }
32 | ">"      { GT }
33 | ">="     { GEQ }
34 | "&&"     { AND }
35 | "||"     { OR }
36 | "|="     { PIPEND }
37 | "!"      { NOT }
38 | "if"     { IF }
39 | "else"   { ELSE }
40 | "for"    { FOR }
```

```
41  | "while"  { WHILE }
42  | "return" { RETURN }
43  | "break"  { BREAK }
44  | "continue" { CONTINUE }
45  | "int"    { INT }
46  | "bool"   { BOOL }
47  | "float"  { FLOAT }
48  | "void"   { VOID }
49  | "char"   { CHAR }
50  | "String" { STRING }
51  | "Point"  { POINT }
52  | "Curve"  { CURVE }
53  | "Canvas" { CANVAS }
54  | "true"   { BLIT(true)  }
55  | "false"  { BLIT(false) }
56  | digits as lxm { LITERAL(int_of_string lxm) }
57  | digits '.'  digit* ( ['e' 'E'] ['+' '-']? digits )? as lxm { FLIT(lxm) }
58  | ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*     as lxm { ID(lxm) }
59  | eof { EOF }
60  | ''' (_ as ch) ''' { CHAR_LITERAL(ch) }
61  | '"' ([^ '"']* as str) '"' { STRING_LITERAL(str) }
62  | _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }
63
64  and comment = parse
65    "*/" { token lexbuf }
66  | _    { comment lexbuf }
67
68  and single = parse
69    '\n' { token lexbuf }
70  | _    { single lexbuf }
```

7.2 Parser

```
1   /* Ocamlyacc parser for SSOL */
2   /*   Authors: Jeevan Farias, Madeleine Tipp, Daniel Mesko */
3
4   %{
5   open Ast
6   %}
7
8   %token SEMI LPAREN RPAREN LBRACE RBRACE COMMA
9   %token LBRACK RBRACK
10  %token PLUS MINUS TIMES MOD DIVIDE ASSIGN
11  %token NOT EQ NEQ LT LEQ GT GEQ AND OR
12  %token DOT PIPE PIPEND
13  %token RETURN IF /*ELIF*/ ELSE FOR WHILE INT BOOL FLOAT VOID
14  %token BREAK CONTINUE
15  %token CHAR STRING POINT CURVE CANVAS
16  %token <int> LITERAL
17  %token <bool> BLIT
18  %token <char> CHAR_LITERAL
19  %token <string> STRING_LITERAL
20  %token <string> ID FLIT
21  %token EOF
22
23  %start program
24  %type <Ast.program> program
25
26  %nonassoc NOELSE
27  %nonassoc ELSE
28  %right ASSIGN
29  %left OR
30  %left AND
31  %left EQ NEQ
32  %left LT GT LEQ GEQ
33  %left PLUS MINUS
34  %left TIMES DIVIDE
35  %left MOD
36  %left PIPE PIPEND
37  %left DOT
38  %right NOT NEG
```

```
39
40 %%
41
42 program:
43   decls EOF { $1 }
44
45 decls:
46    /* nothing */ { ([], [])               }
47  | decls vdecl { (($2 :: fst $1), snd $1) }
48  | decls fdecl { (fst $1, ($2 :: snd $1)) }
49
50 fdecl:
51   typ ID LPAREN formals_opt RPAREN LBRACE stmt_list RBRACE {
52     { typ = $1;
53       fname = $2;
54       formals = List.rev $4;
55       body = List.rev $7 }
56     }
57
58 formals_opt:
59     /* nothing */ { [] }
60   | formal_list   { $1 }
61
62 formal_list:
63     typ ID                   { [($1,$2)]     }
64   | formal_list COMMA typ ID { ($3,$4) :: $1 }
65
66 typ:
67     INT       { Int    }
68   | BOOL      { Bool   }
69   | FLOAT     { Float  }
70   | VOID      { Void   }
71   | CHAR      { Char   }
72   | STRING    { String }
73   | POINT     { Point  }
74   | CURVE     { Curve  }
75   | CANVAS    { Canvas }
76
77 vdecl:
78     typ ID SEMI { ($1, $2) }
79
80 stmt_list:
81     /* nothing */  { [] }
82   | stmt_list stmt { $2 :: $1 }
83
84 vdecl_stmt:
85     typ ID SEMI { VDecl($1,$2)}
86   | typ ID ASSIGN expr SEMI { VDeclAssign($1, $2, $4) }
87   | typ ID ASSIGN array_lit SEMI { VDeclAssign($1, $2, $4) }
88   | typ ID LBRACK LITERAL RBRACK SEMI { ADecl($1, $2, $4) }
89
90 stmt:
91     expr SEMI                                 { Expr $1               }
92   | vdecl_stmt                                { $1                    }
93   | RETURN expr_opt SEMI                      { Return $2             }
94   | LBRACE stmt_list RBRACE                   { Block(List.rev $2)    }
95   | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
96   | IF LPAREN expr RPAREN stmt ELSE stmt     { If($3, $5, $7)        }
97   | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
98                                               { For($3, $5, $7, $9)   }
99   | WHILE LPAREN expr RPAREN stmt             { While($3, $5)         }
100
101 expr_opt:
102     /* nothing */ { Noexpr }
103   | expr          { $1 }
104
105 expr:
106     LITERAL          { Literal($1)            }
107   | FLIT             { Fliteral($1)           }
108   | BLIT             { BoolLit($1)            }
109   | ID               { Id($1)                 }
110   | CHAR_LITERAL     { CharLit($1)            }
```

```
111     | STRING_LITERAL   { StringLit($1)          }
112     | expr PLUS   expr { Binop($1, Add,   $3)   }
113     | expr MINUS  expr { Binop($1, Sub,   $3)   }
114     | expr TIMES  expr { Binop($1, Mult,  $3)   }
115     | expr DIVIDE expr { Binop($1, Div,   $3)   }
116     | expr MOD    expr { Binop($1, Mod,   $3)   }
117     | expr EQ     expr { Binop($1, Equal, $3)   }
118     | expr NEQ    expr { Binop($1, Neq,   $3)   }
119     | expr LT     expr { Binop($1, Less,  $3)   }
120     | expr LEQ    expr { Binop($1, Leq,   $3)   }
121     | expr GT     expr { Binop($1, Greater, $3) }
122     | expr GEQ    expr { Binop($1, Geq,   $3)   }
123     | expr AND    expr { Binop($1, And,   $3)   }
124     | expr OR     expr { Binop($1, Or,    $3)   }
125     | expr PIPEND expr { Binop($1, Pipend, $3)  }
126     | ID   DOT    expr                  { Field($1, $3)                }
127     | MINUS expr %prec NOT              { Unop(Neg, $2)                }
128     | NOT expr                          { Unop(Not, $2)                }
129     | ID ASSIGN expr                    { Assign($1, $3)               }
130     | ID ASSIGN array_lit               { Assign($1, $3)               }
131     | ID LBRACK expr RBRACK             { Access($1, $3)               }
132     | ID LBRACK expr RBRACK ASSIGN expr { ArrayAssign($1, $3, $6)      }
133     | ID LPAREN args_opt RPAREN         { Call($1, $3)                 }
134     | typ LPAREN args_opt RPAREN        { Call((string_of_typ $1), $3) }
135     | LPAREN expr RPAREN                { $2 }
136
137   array_lit:
138     LBRACE args_opt RBRACE { ArrayLit($2) }
139
140   args_opt:
141       /* nothing */ { [] }
142     | args_list  { List.rev $1 }
143
144   args_list:
145       expr                    { [$1] }
146     | args_list COMMA expr    { $3 :: $1 }
```

7.3 AST

```
1   (* SSOL Abstract Syntax Tree and functions for printing it
2       Authors: Jeevan Farias, Madeleine Tipp, Daniel Mesko *)
3
4   type op = Add | Sub | Mult | Div | Mod | Equal | Neq | Less | Leq | Greater | Geq |
5             And | Or | Pipend
6
7   type uop = Neg | Not
8
9   type typ = Int | Bool | Float | Void | Char | String | Point | Curve | Canvas
10                | Array of typ * int
11
12  type bind = typ * string
13
14  type expr =
15      Literal of int
16    | Fliteral of string
17    | BoolLit of bool
18    | CharLit of char
19    | StringLit of string
20    | ArrayLit of expr list
21    | Id of string
22    | Binop of expr * op * expr
23    | Field of string * expr
24    | Unop of uop * expr
25    | Assign of string * expr
26    | Access of string * expr
27    | ArrayAssign of string * expr * expr
28    | Call of string * expr list
29    | Noexpr
30
31  type stmt =
32      VDecl of typ * string
```

```
33    | VDeclAssign of typ * string * expr
34    | ADecl of typ * string * int
35    | Block of stmt list
36    | Expr of expr
37    | Return of expr
38    | If of expr * stmt * stmt
39    | For of expr * expr * expr * stmt
40    | While of expr * stmt
41
42  type func_decl = {
43      typ : typ;
44      fname : string;
45      formals : bind list;
46      body : stmt list;
47    }
48
49  type program = bind list * func_decl list
50
51  (* Pretty-printing functions *)
52
53  let string_of_op = function
54      Add -> "+"
55    | Sub -> "-"
56    | Mult -> "*"
57    | Div -> "/"
58    | Mod -> "%"
59    | Equal -> "=="
60    | Neq -> "!="
61    | Less -> "<"
62    | Leq -> "<="
63    | Greater -> ">"
64    | Geq -> ">="
65    | And -> "&&"
66    | Or -> "||"
67    | Pipend -> "|="
68
69  let string_of_uop = function
70      Neg -> "-"
71    | Not -> "!"
72
73  let rec string_of_typ = function
74      Int -> "int"
75    | Bool -> "bool"
76    | Float -> "float"
77    | Void -> "void"
78    | Char -> "char"
79    | String -> "String"
80    | Point -> "Point"
81    | Curve -> "Curve"
82    | Canvas -> "Canvas"
83    | Array(t, n) -> (string_of_typ t) ^ "[" ^ (string_of_int n) ^ "]"
84
85  let rec string_of_expr = function
86      Literal(l) -> string_of_int l
87    | Fliteral(l) -> l
88    | BoolLit(true) -> "true"
89    | BoolLit(false) -> "false"
90    | CharLit(l) -> String.make 1 l
91    | StringLit(l) -> l
92    | ArrayLit(arr) -> "[" ^ (List.fold_left (fun lst elem -> lst ^ " " ^ string_of_expr
          elem ^ ",") "" arr) ^ "]"
93    | Field(s,f) -> s ^ "." ^ string_of_expr f
94    | Id(s) -> s
95    | Binop(e1, o, e2) ->
96        string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
97    | Unop(o, e) -> string_of_uop o ^ string_of_expr e
98    | Assign(v, e) -> v ^ " = " ^ string_of_expr e
99    | Access(a, i) -> a ^ "[" ^ string_of_expr i ^ "]"
100   | ArrayAssign(arr, index, rval) -> arr ^ "[" ^ string_of_expr index ^ "]" ^
          string_of_expr rval
101   | Call(f, el) ->
102       f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
```

```
103     | Noexpr -> ""
104
105     let rec string_of_stmt = function
106       VDecl(t, i) -> string_of_typ t ^ " " ^ i ^ "\n"
107     | VDeclAssign(t, i, e) -> string_of_typ t ^ " " ^ i ^ " = " ^ string_of_expr e ^ "\n
          "
108     | ADecl(t, i, s) -> string_of_typ t ^ " " ^ i ^ "[" ^ string_of_int s ^ "]\n"
109     | Block(stmts) ->
110         "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
111     | Expr(expr) -> string_of_expr expr ^ ";\n";
112     | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
113     | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt s
114     | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
115         string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
116     | For(e1, e2, e3, s) ->
117         "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ; " ^
118         string_of_expr e3  ^ ") " ^ string_of_stmt s
119     | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s
120
121
122   let string_of_vdecl (t, id) = string_of_typ t ^ " " ^ id ^ ";\n"
123
124   let string_of_fdecl fdecl =
125     string_of_typ fdecl.typ ^ " " ^
126     fdecl.fname ^ "(" ^ String.concat ", " (List.map snd fdecl.formals) ^
127     ")\n{\n" ^
128     (* String.concat "" (List.map string_of_vdecl fdecl.locals) ^ *)
129     String.concat "" (List.map string_of_stmt fdecl.body) ^
130     "}\n"
131
132   let string_of_program (vars, funcs) =
133     String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
134     String.concat "\n" (List.map string_of_fdecl funcs)
```

7.4 Semantic Checker

```
1   (* Semantic checking for the SSOL compiler *)
2   (*     Authors: Jeevan Farias, Madeleine Tipp, Daniel Mesko *)
3
4   open Ast
5   open Sast
6
7   module StringMap = Map.Make(String)
8
9   (* Semantic checking of the AST. Returns an SAST if successful,
10     throws an exception if something is wrong.
11
12     Check each global variable, then check each function *)
13
14  let check (globals, functions) =
15
16    (* Verify a list of bindings has no void types or duplicate names *)
17    let check_binds (kind : string) (binds : bind list) =
18      List.iter (function
19    (Void, b) -> raise (Failure ("illegal void " ^ kind ^ " " ^ b))
20        | _ -> ()) binds;
21      let rec dups = function
22          [] -> ()
23        | ((_,n1) :: (_,n2) :: _) when n1 = n2 ->
24      raise (Failure ("duplicate " ^ kind ^ " " ^ n1))
25        | _ :: t -> dups t
26      in dups (List.sort (fun (_,a) (_,b) -> compare a b) binds)
27    in
28
29    (**** Check global variables ****)
30
31    check_binds "global" globals;
32
33    (**** Check functions ****)
34
35    (* Collect function declarations for built-in functions: no bodies *)
```

```
36
37    let built_in_decls =
38      let add_bind map (name, retyp, formlist) = StringMap.add name {
39        typ = retyp;
40        fname = name;
41        formals = formlist;
42        (* locals = []; *) body = [] } map
43      in List.fold_left add_bind StringMap.empty [ ("print", Void, [(Int, "x")]);
44                           ("printb", Void, [(Bool, "x")]);
45                           ("printf", Void, [(Float, "x")]);
46                           ("printbig", Void, [(Int, "x")]);
47                           ("prints", Void, [(String, "x")]);
48                           ("draw", Void, [(Canvas, "can"); (String, "filename")]);
49                           ("Point", Point, [(Float, "x"); (Float, "y")]);
50                           ("Curve", Curve, [(Point, "ep1") ; (Point, "ep2") ;
51                             (Point, "cp1") ; (Point, "cp2")]);
52                           ("Canvas", Canvas, [(Float, "x"); (Float, "y")]);]
53
54    in
55
56    (* Add function name to symbol table *)
57    let add_func map fd =
58      let built_in_err = "function " ^ fd.fname ^ " may not be defined"
59      and dup_err = "duplicate function " ^ fd.fname
60      and make_err er = raise (Failure er)
61      and n = fd.fname (* Name of the function *)
62      in match fd with (* No duplicate functions or redefinitions of built-ins *)
63           _ when StringMap.mem n built_in_decls -> make_err built_in_err
64         | _ when StringMap.mem n map -> make_err dup_err
65         | _ ->  StringMap.add n fd map
66    in
67
68    (* Collect all function names into one symbol table *)
69    let function_decls = List.fold_left add_func built_in_decls functions
70    in
71
72    (* Return a function from our symbol table *)
73    let find_func s =
74      try StringMap.find s function_decls
75      with Not_found -> raise (Failure ("unrecognized function " ^ s))
76    in
77
78    let _ = find_func "main" in (* Ensure "main" is defined *)
79
80    let check_function func =
81      (* Make sure no formals or locals are void or duplicates *)
82      check_binds "formal" func.formals;
83      (* check_binds "local" func.locals; *)
84
85
86    (* Raise an exception if the given rvalue type cannot be assigned to
87         the given lvalue type *)
88    let check_assign lvaluet rvaluet err =
89       match lvaluet with
90             Array(lt, ls) ->
91                (match rvaluet with
92                     Array(rt, rs) -> if (lt == rt && ls == rs) then lvaluet else raise (
                         Failure err)
93                   | _ -> raise (Failure err))
94           | _ -> if lvaluet == rvaluet then lvaluet else raise (Failure err)
95      in
96
97
98      (* Build initial symbol table with globals and formals *)
99      let globmap = List.fold_left (fun m (ty, name) -> StringMap.add name ty m)
100         StringMap.empty (globals @ func.formals)
101     in
102
103     (* Return type of a symbol from supplied symbol table *)
104     let type_of_identifier locals s =
105       try StringMap.find s locals
106       with Not_found -> raise (Failure ("undeclared identifier " ^ s))
```

```
107     in
108
109     (* Return member symbol map for a particular type *)
110     let member_map_of_type ty = match ty with
111         Point | Canvas -> List.fold_left (fun m (ty, name) -> StringMap.add name ty m)
112             StringMap.empty [(Float, "x"); (Float, "y")]
113       | Curve -> List.fold_left (fun m (ty, name) -> StringMap.add name ty m)
114             StringMap.empty [(Point, "ep1"); (Point, "ep2"); (Point, "cp1");
115             (Point, "cp2")]
116       | _ -> raise (Failure ("type " ^ string_of_typ ty ^ " does not have members"))
117
118      in
119
120
121     (* Return a semantically-checked expression, i.e., with a type *)
122
123     let rec expr locals = function
124         Literal  l  -> (Int, SLiteral l)
125       | Fliteral l  -> (Float, SFliteral l)
126       | BoolLit l   -> (Bool, SBoolLit l)
127       | CharLit l   -> (Char, SCharLit l)
128       | StringLit l -> (String, SStringLit l)
129       | ArrayLit elist ->
130           let typmatch t (ty, _) =
131             if t == ty then
132               ty
133             else
134               raise (Failure ("array elements are not of same type"))
135           and slist = List.map (fun e -> expr locals e) elist in
136           (match slist with
137               [] -> raise (Failure "cannot have array literal with 0 elements")
138             | _ ->
139               let ty = List.fold_left typmatch (fst (List.hd slist)) slist in
140               (Array(ty, List.length slist), SArrayLit(slist)) )
141       | Noexpr      -> (Void, SNoexpr)
142       | Id s        -> (type_of_identifier locals s, SId s)
143       | Assign(var, e) as ex ->
144           let lt = type_of_identifier locals var
145           and (rt, e') = expr locals e in
146           let err = "illegal assignment " ^ string_of_typ lt ^ " = " ^
147             string_of_typ rt ^ " in " ^ string_of_expr ex ^ " for identifier " ^ var
148           in (check_assign lt rt err, SAssign(var, (rt, e')))
149       | Access(arr, ind) ->
150           let arrtyp = type_of_identifier locals arr
151           and (ityp, _) as ind' = expr locals ind in
152           (match arrtyp with
153               Array(t, _) -> (match ityp with
154                   Int -> (t, SAccess(arr, ind'))
155                 | _ -> raise (Failure ("expected Int for array index value, " ^
156                         "but was given " ^ string_of_sexpr ind')) )
157             | _ -> raise (Failure ("cannot access index " ^ string_of_sexpr ind' ^
158                 " of " ^ arr ^ ": it has type " ^ string_of_typ arrtyp)) )
159       | ArrayAssign(arr, ind, ex) ->
160            let arrtyp = type_of_identifier locals arr
161           and ind' = expr locals ind
162           and ex'= expr locals ex in
163           let err = "illegal assignment " ^ (string_of_typ arrtyp) ^
164               " = " ^ (string_of_typ (fst ex')) in
165           (match arrtyp with
166               Array(t, _) -> (check_assign t (fst ex') err,
167                    SArrayAssign(arr, ind', ex'))
168             | _ -> raise (Failure (err)) )
169       | Field(obj, mem)  ->
170           let ty = type_of_identifier locals obj in
171           let memmap = member_map_of_type ty in
172           let smem = match mem with
173               Assign(v,e) as ex->
174                    let ty = type_of_identifier memmap v in
175                       (match e with
176                          Fliteral _ ->
177                             let lt = StringMap.find v memmap
178                             and (rt, e') = expr locals e in
```

```
179                             let err = "illegal assignment of object field" ^
180                                 string_of_typ lt ^ " = " ^
181                                 string_of_typ rt ^ " in " ^
182                                 string_of_expr ex ^ " for identifier Field." ^ v
183                             in (check_assign lt rt err, SAssign(v, (rt, e')))
184                         | Id s ->  (ty,SAssign(v,(ty, SId s)) )
185                         | _ -> raise (Failure ("illegal member access - "
186                                 ^ " expression type is not a field")))
187               | _ -> expr memmap mem
188             in
189           (fst smem, SField(obj, smem))
190
191       | Unop(op, e) as ex ->
192           let (t, e') = expr locals e in
193           let ty = match op with
194             Neg when t = Int || t = Float -> t
195           | Not when t = Bool -> Bool
196           | _ -> raise (Failure ("illegal unary operator " ^
197                                  string_of_uop op ^ string_of_typ t ^
198                                  " in " ^ string_of_expr ex))
199           in (ty, SUnop(op, (t, e')))
200       | Binop(e1, op, e2) as e ->
201           let (t1, e1') = expr locals e1
202           and (t2, e2') = expr locals e2 in
203           (* All binary operators require operands of the same type *)
204           let same = t1 = t2 in
205           (* Determine expression type based on operator and operand types *)
206           let ty = match op with
207             Add | Sub | Mult | Div when same && t1 = Int   -> Int
208           | Add | Sub | Mult | Div when same && t1 = Float -> Float
209           | Equal | Neq            when same               -> Bool
210           | Less | Leq | Greater | Geq
211                       when same && (t1 = Int || t1 = Float) -> Bool
212           | And | Or when same && t1 = Bool -> Bool
213           | Mod when same && t1 = Int -> Int
214           | Pipend when t1 = Canvas && t2 = Curve -> Canvas
215           | _ -> raise (
216         Failure ("illegal binary operator " ^
217                       string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
218                       string_of_typ t2 ^ " in " ^ string_of_expr e))
219           in (ty, SBinop((t1, e1'), op, (t2, e2')))
220       | Call(fname, args) as call ->
221           let fd = find_func fname in
222           let param_length = List.length fd.formals in
223           if List.length args != param_length then
224             raise (Failure ("expecting " ^ string_of_int param_length ^
225                             " arguments in " ^ string_of_expr call))
226           else let check_call (ft, _) e =
227             let (et, e') = expr locals e in
228             let err = "illegal argument found " ^ string_of_typ et ^
229               " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr e
230             in (check_assign ft et err, e')
231           in
232           let args' = List.map2 check_call fd.formals args
233           in (fd.typ, SCall(fname, args'))
234     in
235
236     let check_bool_expr locals e =
237       let (t', e') = expr locals e
238       and err = "expected Boolean expression in " ^ string_of_expr e
239       in if t' != Bool then raise (Failure err) else (t', e')
240     in
241
242     (* Return a semantically-checked statement i.e. containing sexprs *)
243
244     let rec check_stmt locals = function
245       Block sl ->
246         let rec check_block block_locals ssl= function
247         [Return _ as s] -> ssl @ [check_stmt block_locals s]
248           | Return _::_ -> raise (Failure "nothing may follow a return")
249           | Block sl :: ss -> [check_stmt block_locals (Block sl)]
250                               @ (check_block block_locals ssl ss)
```

```
251           | s :: ss ->
252             (match s with
253                 VDecl(t,name) ->
254                   (match t with
255              Void -> raise(Failure ("illegal void local "^name))
256             | _ -> let block_locals = StringMap.add name t block_locals
257                           in [check_stmt block_locals s] @ check_block block_locals
                                  ssl ss)
258               | VDeclAssign(t,name,e) ->
259         if t == Void then raise(Failure ("illegal void local "^name) )
260         else
261                 let sx = expr block_locals e in
262                 let typ = (match fst(sx) with
263                         Array(tp,s) -> if tp == t
264                           then Array(tp,s)
265                           else raise(Failure("Array literal is of inconsistent type"))
266                       | _ -> if fst(sx) == t
267                           then fst(sx)
268                           else raise(Failure("illegal assignment"))) in
269                 let block_locals = StringMap.add name typ block_locals in
270                   [check_stmt block_locals s] @ check_block block_locals ssl ss
271               | ADecl(t,name, n) ->
272         if t == Void then raise(Failure ("illegal void local "^name))
273         else
274                 let block_locals = StringMap.add name (Array(t,n)) block_locals
275                   in [check_stmt block_locals s] @ check_block block_locals ssl ss
276               | _ -> [check_stmt block_locals s] @ check_block block_locals ssl ss)
277           | [] -> ssl
278         in SBlock(check_block locals [] sl)
279     | VDecl(t,s) -> SVDecl(t,s)
280     | VDeclAssign(_,s,e) ->
281       let sx = expr locals e in
282       let ty = type_of_identifier locals s in
283       SVDeclAssign(ty,s,sx)
284     | ADecl(t,s,n) -> SADecl(t,s,n)
285     | Expr e -> SExpr (expr locals e)
286     | If(p, b1, b2) -> SIf(check_bool_expr locals p, check_stmt locals b1,
287                            check_stmt locals b2)
288     | For(e1, e2, e3, st) ->
289         SFor(expr locals e1, check_bool_expr locals e2, expr locals e3,
290              check_stmt locals st)
291     | While(p, s) -> SWhile(check_bool_expr locals p, check_stmt locals s)
292     | Return e -> let (t, e') = expr locals e in
293         if t = func.typ then SReturn (t, e')
294         else raise (Failure ("return gives " ^ string_of_typ t ^ " expected " ^
295        string_of_typ func.typ ^ " in " ^ string_of_expr e))
296     in (* body of check_function *)
297     { styp = func.typ;
298       sfname = func.fname;
299       sformals = func.formals;
300       sbody = match check_stmt globmap (Block func.body) with
301          SBlock(sl) -> sl
302       | _ -> raise (Failure ("internal error: block didn't become a block?"))
303     }
304   in (globals, List.map check_function functions)
```

7.5 SAST

```
1  (* Semantically-checked Abstract Syntax Tree and functions for printing it *)
2  (*   Authors: Jeevan Farias, Madeleine Tipp, Daniel Mesko *)
3
4  open Ast
5
6  type sexpr = typ * sx
7  and sx =
8      SLiteral of int
9    | SFliteral of string
10   | SBoolLit of bool
11   | SCharLit of char
12   | SStringLit of string
13   | SArrayLit of sexpr list
```

```
14    | SId of string
15    | SBinop of sexpr * op * sexpr
16    | SField of string * sexpr
17    | SUnop of uop * sexpr
18    | SAssign of string * sexpr
19    | SAccess of string * sexpr
20    | SArrayAssign of string * sexpr * sexpr
21    | SCall of string * sexpr list
22    | SNoexpr
23
24  type sstmt =
25      SVDecl of typ * string
26    | SVDeclAssign of typ * string * sexpr
27    | SADecl of typ * string * int
28    | SBlock of sstmt list
29    | SExpr of sexpr
30    | SReturn of sexpr
31    | SIf of sexpr * sstmt * sstmt
32    | SFor of sexpr * sexpr * sexpr * sstmt
33    | SWhile of sexpr * sstmt
34
35  type sfunc_decl = {
36      styp : typ;
37      sfname : string;
38      sformals : bind list;
39      sbody : sstmt list;
40    }
41
42  type sprogram = bind list * sfunc_decl list
43
44  (* Pretty-printing functions *)
45
46  let rec string_of_sexpr (t, e) =
47    "(" ^ string_of_typ t ^ " : " ^ (match e with
48      SLiteral(l) -> string_of_int l
49    | SFliteral(l) -> l
50    | SBoolLit(true) -> "true"
51    | SBoolLit(false) -> "false"
52    | SCharLit(l) -> String.make 1 l
53    | SStringLit(l) -> l
54    | SArrayLit(arr) -> "[" ^ (List.fold_left (fun lst elem -> lst ^ " " ^
        string_of_sexpr elem ^ ",") "" arr) ^ "]"
55    | SField(e,f) -> e ^ "." ^ string_of_sexpr f
56    | SId(s) -> s
57    | SBinop(e1, o, e2) ->
        string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_sexpr e2
58    | SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
59    | SAssign(v, e) -> v ^ " = " ^ string_of_sexpr e
60    | SAccess(a, i) -> a ^ "[" ^ string_of_sexpr i ^ "]"
61    | SArrayAssign(arr, index, rval) -> arr ^ "[" ^ string_of_sexpr index ^ "]" ^
        string_of_sexpr rval
62    | SCall(f, el) ->
        f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^ ")"
63    | SNoexpr -> ""  ) ^ ")"
64
65  let rec string_of_sstmt = function
66      SVDecl(t, i) -> string_of_typ t ^ " " ^ i ^ "\n"
67    | SVDeclAssign(t, i, e) -> string_of_typ t ^ " " ^ i ^ " = " ^ string_of_sexpr e ^
        "\n"
68    | SADecl(t, i, s) -> string_of_typ t ^ " " ^ i ^ "[" ^ string_of_int s ^ "]\n"
69    | SBlock(stmts) ->
        "{\n" ^ String.concat "" (List.map string_of_sstmt stmts) ^ "}\n"
70    | SExpr(expr) -> string_of_sexpr expr ^ ";\n";
71    | SReturn(expr) -> "return " ^ string_of_sexpr expr ^ ";\n";
72    | SIf(e, s, SBlock([])) ->
        "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s
73    | SIf(e, s1, s2) ->  "if (" ^ string_of_sexpr e ^ ")\n" ^
        string_of_sstmt s1 ^ "else\n" ^ string_of_sstmt s2
74    | SFor(e1, e2, e3, s) ->
        "for (" ^ string_of_sexpr e1  ^ " ; " ^ string_of_sexpr e2 ^ " ; " ^
        string_of_sexpr e3  ^ ") " ^ string_of_sstmt s
75    | SWhile(e, s) -> "while (" ^ string_of_sexpr e ^ ") " ^ string_of_sstmt s
```

```
83
84  let string_of_sfdecl fdecl =
85    string_of_typ fdecl.styp ^ " " ^
86    fdecl.sfname ^ "(" ^ String.concat ", " (List.map snd fdecl.sformals) ^
87    ")\n{\n" ^
88    (* String.concat "" (List.map string_of_vdecl fdecl.slocals) ^ *)
89    String.concat "" (List.map string_of_sstmt fdecl.sbody) ^
90    "}\n"
91
92  let string_of_sprogram (vars, funcs) =
93    String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
94    String.concat "\n" (List.map string_of_sfdecl funcs)
```

7.6 Code Generator

```
1   (* Semantic checking for the SSOL compiler *)
2   (*    Authors: Jeevan Farias, Madeleine Tipp, Daniel Mesko *)
3
4   open Ast
5   open Sast
6
7   module StringMap = Map.Make(String)
8
9   (* Semantic checking of the AST. Returns an SAST if successful,
10     throws an exception if something is wrong.
11
12     Check each global variable, then check each function *)
13
14  let check (globals, functions) =
15
16    (* Verify a list of bindings has no void types or duplicate names *)
17    let check_binds (kind : string) (binds : bind list) =
18      List.iter (function
19    (Void, b) -> raise (Failure ("illegal void " ^ kind ^ " " ^ b))
20        | _ -> ()) binds;
21      let rec dups = function
22          [] -> ()
23        | ((_,n1) :: (_,n2) :: _) when n1 = n2 ->
24      raise (Failure ("duplicate " ^ kind ^ " " ^ n1))
25        | _ :: t -> dups t
26      in dups (List.sort (fun (_,a) (_,b) -> compare a b) binds)
27    in
28
29    (**** Check global variables ****)
30
31    check_binds "global" globals;
32
33    (**** Check functions ****)
34
35    (* Collect function declarations for built-in functions: no bodies *)
36
37    let built_in_decls =
38      let add_bind map (name, retyp, formlist) = StringMap.add name {
39        typ = retyp;
40        fname = name;
41        formals = formlist;
42        (* locals = []; *) body = [] } map
43      in List.fold_left add_bind StringMap.empty [ ("print", Void, [(Int, "x")]);
44                           ("printb", Void, [(Bool, "x")]);
45                           ("printf", Void, [(Float, "x")]);
46                           ("printbig", Void, [(Int, "x")]);
47                           ("prints", Void, [(String, "x")]);
48                           ("draw", Void, [(Canvas, "can"); (String, "filename")]);
49                           ("Point", Point, [(Float, "x"); (Float, "y")]);
50                           ("Curve", Curve, [(Point, "ep1") ; (Point, "ep2") ;
51                             (Point, "cp1") ; (Point, "cp2")]);
52                           ("Canvas", Canvas, [(Float, "x"); (Float, "y")]);]
53
54    in
55
56    (* Add function name to symbol table *)
```

```
57    let add_func map fd =
58      let built_in_err = "function " ^ fd.fname ^ " may not be defined"
59      and dup_err = "duplicate function " ^ fd.fname
60      and make_err er = raise (Failure er)
61      and n = fd.fname (* Name of the function *)
62      in match fd with (* No duplicate functions or redefinitions of built-ins *)
63           _ when StringMap.mem n built_in_decls -> make_err built_in_err
64         | _ when StringMap.mem n map -> make_err dup_err
65         | _ ->  StringMap.add n fd map
66    in
67
68    (* Collect all function names into one symbol table *)
69    let function_decls = List.fold_left add_func built_in_decls functions
70    in
71
72    (* Return a function from our symbol table *)
73    let find_func s =
74      try StringMap.find s function_decls
75      with Not_found -> raise (Failure ("unrecognized function " ^ s))
76    in
77
78    let _ = find_func "main" in (* Ensure "main" is defined *)
79
80    let check_function func =
81      (* Make sure no formals or locals are void or duplicates *)
82      check_binds "formal" func.formals;
83      (* check_binds "local" func.locals; *)
84
85
86    (* Raise an exception if the given rvalue type cannot be assigned to
87       the given lvalue type *)
88    let check_assign lvaluet rvaluet err =
89       match lvaluet with
90             Array(lt, ls) ->
91                (match rvaluet with
92                     Array(rt, rs) -> if (lt == rt && ls == rs) then lvaluet else raise (
                             Failure err)
93                   | _ -> raise (Failure err))
94           | _ -> if lvaluet == rvaluet then lvaluet else raise (Failure err)
95      in
96
97
98      (* Build initial symbol table with globals and formals *)
99      let globmap = List.fold_left (fun m (ty, name) -> StringMap.add name ty m)
100         StringMap.empty (globals @ func.formals)
101     in
102
103     (* Return type of a symbol from supplied symbol table *)
104     let type_of_identifier locals s =
105       try StringMap.find s locals
106       with Not_found -> raise (Failure ("undeclared identifier " ^ s))
107     in
108
109     (* Return member symbol map for a particular type *)
110     let member_map_of_type ty = match ty with
111         Point | Canvas -> List.fold_left (fun m (ty, name) -> StringMap.add name ty m)
112             StringMap.empty [(Float, "x"); (Float, "y")]
113       | Curve -> List.fold_left (fun m (ty, name) -> StringMap.add name ty m)
114             StringMap.empty [(Point, "ep1"); (Point, "ep2"); (Point, "cp1");
115             (Point, "cp2")]
116       | _ -> raise (Failure ("type " ^ string_of_typ ty ^ " does not have members"))
117
118      in
119
120
121     (* Return a semantically-checked expression, i.e., with a type *)
122
123     let rec expr locals = function
124         Literal  l  -> (Int, SLiteral l)
125       | Fliteral l  -> (Float, SFliteral l)
126       | BoolLit l   -> (Bool, SBoolLit l)
127       | CharLit l   -> (Char, SCharLit l)
```

```
128         | StringLit l -> (String, SStringLit l)
129         | ArrayLit elist ->
130             let typmatch t (ty, _) =
131               if t == ty then
132                 ty
133               else
134                 raise (Failure ("array elements are not of same type"))
135             and slist = List.map (fun e -> expr locals e) elist in
136             (match slist with
137                 [] -> raise (Failure "cannot have array literal with 0 elements")
138               | _ ->
139                 let ty = List.fold_left typmatch (fst (List.hd slist)) slist in
140                 (Array(ty, List.length slist), SArrayLit(slist)) )
141         | Noexpr      -> (Void, SNoexpr)
142         | Id s        -> (type_of_identifier locals s, SId s)
143         | Assign(var, e) as ex ->
144             let lt = type_of_identifier locals var
145             and (rt, e') = expr locals e in
146             let err = "illegal assignment " ^ string_of_typ lt ^ " = " ^
147               string_of_typ rt ^ " in " ^ string_of_expr ex ^ " for identifier " ^ var
148             in (check_assign lt rt err, SAssign(var, (rt, e')))
149         | Access(arr, ind) ->
150             let arrtyp = type_of_identifier locals arr
151             and (ityp, _) as ind' = expr locals ind in
152             (match arrtyp with
153                 Array(t, _) -> (match ityp with
154                     Int -> (t, SAccess(arr, ind'))
155                   | _ -> raise (Failure ("expected Int for array index value, " ^
156                                 "but was given " ^ string_of_sexpr ind')) )
157               | _ -> raise (Failure ("cannot access index " ^ string_of_sexpr ind' ^
158                     " of " ^ arr ^ ": it has type " ^ string_of_typ arrtyp)) )
159         | ArrayAssign(arr, ind, ex) ->
160              let arrtyp = type_of_identifier locals arr
161             and ind' = expr locals ind
162             and ex'= expr locals ex in
163             let err = "illegal assignment " ^ (string_of_typ arrtyp) ^
164                 " = " ^ (string_of_typ (fst ex')) in
165             (match arrtyp with
166                 Array(t, _) -> (check_assign t (fst ex') err,
167                      SArrayAssign(arr, ind', ex'))
168               | _ -> raise (Failure (err)) )
169         | Field(obj, mem)  ->
170             let ty = type_of_identifier locals obj in
171             let memmap = member_map_of_type ty in
172             let smem = match mem with
173                 Assign(v,e) as ex->
174                      let ty = type_of_identifier memmap v in
175                         (match e with
176                            Fliteral _ ->
177                              let lt = StringMap.find v memmap
178                              and (rt, e') = expr locals e in
179                              let err = "illegal assignment of object field" ^
180                                  string_of_typ lt ^ " = " ^
181                                  string_of_typ rt ^ " in " ^
182                                  string_of_expr ex ^ " for identifier Field." ^ v
183                              in (check_assign lt rt err, SAssign(v, (rt, e')))
184                            | Id s ->  (ty,SAssign(v,(ty, SId s)) )
185                            | _ -> raise (Failure ("illegal member access - "
186                                  ^ " expression type is not a field")))
187                 | _ -> expr memmap mem
188               in
189             (fst smem, SField(obj, smem))
190
191         | Unop(op, e) as ex ->
192             let (t, e') = expr locals e in
193             let ty = match op with
194               Neg when t = Int || t = Float -> t
195             | Not when t = Bool -> Bool
196             | _ -> raise (Failure ("illegal unary operator " ^
197                                    string_of_uop op ^ string_of_typ t ^
198                                    " in " ^ string_of_expr ex))
199             in (ty, SUnop(op, (t, e')))
```

```
200       | Binop(e1, op, e2) as e ->
201           let (t1, e1') = expr locals e1
202           and (t2, e2') = expr locals e2 in
203           (* All binary operators require operands of the same type *)
204           let same = t1 = t2 in
205           (* Determine expression type based on operator and operand types *)
206           let ty = match op with
207             Add | Sub | Mult | Div when same && t1 = Int   -> Int
208           | Add | Sub | Mult | Div when same && t1 = Float -> Float
209           | Equal | Neq            when same               -> Bool
210           | Less | Leq | Greater | Geq
211                       when same && (t1 = Int || t1 = Float) -> Bool
212           | And | Or when same && t1 = Bool -> Bool
213           | Mod when same && t1 = Int -> Int
214           | Pipend when t1 = Canvas && t2 = Curve -> Canvas
215           | _ -> raise (
216         Failure ("illegal binary operator " ^
217                       string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
218                       string_of_typ t2 ^ " in " ^ string_of_expr e))
219         in (ty, SBinop((t1, e1'), op, (t2, e2')))
220       | Call(fname, args) as call ->
221           let fd = find_func fname in
222           let param_length = List.length fd.formals in
223           if List.length args != param_length then
224             raise (Failure ("expecting " ^ string_of_int param_length ^
225                             " arguments in " ^ string_of_expr call))
226           else let check_call (ft, _) e =
227             let (et, e') = expr locals e in
228             let err = "illegal argument found " ^ string_of_typ et ^
229               " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr e
230             in (check_assign ft et err, e')
231           in
232           let args' = List.map2 check_call fd.formals args
233           in (fd.typ, SCall(fname, args'))
234     in
235
236     let check_bool_expr locals e =
237       let (t', e') = expr locals e
238       and err = "expected Boolean expression in " ^ string_of_expr e
239       in if t' != Bool then raise (Failure err) else (t', e')
240     in
241
242     (* Return a semantically-checked statement i.e. containing sexprs *)
243
244     let rec check_stmt locals = function
245       Block sl ->
246         let rec check_block block_locals ssl= function
247         [Return _ as s] -> ssl @ [check_stmt block_locals s]
248           | Return _::_ -> raise (Failure "nothing may follow a return")
249           | Block sl :: ss -> [check_stmt block_locals (Block sl)]
250                               @ (check_block block_locals ssl ss)
251           | s :: ss ->
252             (match s with
253                 VDecl(t,name) ->
254                   (match t with
255              Void -> raise(Failure ("illegal void local "^name))
256           | _ -> let block_locals = StringMap.add name t block_locals
257                                 in [check_stmt block_locals s] @ check_block block_locals
                                       ssl ss)
258               | VDeclAssign(t,name,e) ->
259         if t == Void then raise(Failure ("illegal void local "^name) )
260         else
261                 let sx = expr block_locals e in
262                 let typ = (match fst(sx) with
263                         Array(tp,s) -> if tp == t
264                           then Array(tp,s)
265                           else raise(Failure("Array literal is of inconsistent type"))
266                       | _ -> if fst(sx) == t
267                           then fst(sx)
268                           else raise(Failure("illegal assignment"))) in
269                 let block_locals = StringMap.add name typ block_locals in
270                   [check_stmt block_locals s] @ check_block block_locals ssl ss
```

```
271                 | ADecl(t,name, n) ->
272           if t == Void then raise(Failure ("illegal void local "^name))
273           else
274                   let block_locals = StringMap.add name (Array(t,n)) block_locals
275                     in [check_stmt block_locals s] @ check_block block_locals ssl ss
276                 | _ -> [check_stmt block_locals s] @ check_block block_locals ssl ss)
277             | [] -> ssl
278           in SBlock(check_block locals [] sl)
279       | VDecl(t,s) -> SVDecl(t,s)
280       | VDeclAssign(_,s,e) ->
281         let sx = expr locals e in
282         let ty = type_of_identifier locals s in
283         SVDeclAssign(ty,s,sx)
284       | ADecl(t,s,n) -> SADecl(t,s,n)
285       | Expr e -> SExpr (expr locals e)
286       | If(p, b1, b2) -> SIf(check_bool_expr locals p, check_stmt locals b1,
287                              check_stmt locals b2)
288       | For(e1, e2, e3, st) ->
289           SFor(expr locals e1, check_bool_expr locals e2, expr locals e3,
290                check_stmt locals st)
291       | While(p, s) -> SWhile(check_bool_expr locals p, check_stmt locals s)
292       | Return e -> let (t, e') = expr locals e in
293         if t = func.typ then SReturn (t, e')
294         else raise (Failure ("return gives " ^ string_of_typ t ^ " expected " ^
295          string_of_typ func.typ ^ " in " ^ string_of_expr e))
296       in (* body of check_function *)
297       { styp = func.typ;
298         sfname = func.fname;
299         sformals = func.formals;
300         sbody = match check_stmt globmap (Block func.body) with
301           SBlock(sl) -> sl
302         | _ -> raise (Failure ("internal error: block didn't become a block?"))
303       }
304     in (globals, List.map check_function functions)
```

7.7 SSOL - top-level

```
1  (* Top-level of the SSOL compiler: scan & parse the input,
2     check the resulting AST and generate an SAST from it, generate LLVM IR,
3     and dump the module
4
5     Authors: Jeevan Farias, Madeleine Tipp, Daniel Mesko *)
6
7  type action = Ast | Sast | LLVM_IR | Compile
8
9  let () =
10   let action = ref Compile in
11   let set_action a () = action := a in
12   let speclist = [
13     ("-a", Arg.Unit (set_action Ast), "Print the AST");
14     ("-s", Arg.Unit (set_action Sast), "Print the SAST");
15     ("-l", Arg.Unit (set_action LLVM_IR), "Print the generated LLVM IR");
16     ("-c", Arg.Unit (set_action Compile),
17       "Check and print the generated LLVM IR (default)");
18   ] in
19   let usage_msg = "usage: ./microc.native [-a|-s|-l|-c] [file.mc]" in
20   let channel = ref stdin in
21   Arg.parse speclist (fun filename -> channel := open_in filename) usage_msg;
22
23   let lexbuf = Lexing.from_channel !channel in
24   let ast = Parserssol.program Scanner.token lexbuf in
25   match !action with
26     Ast -> print_string (Ast.string_of_program ast)
27   | _ -> let sast = Semant.check ast in
28     match !action with
29       Ast     -> ()
30     | Sast    -> print_string (Sast.string_of_sprogram sast)
31     | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate sast))
32     | Compile -> let m = Codegen.translate sast in
33  Llvm_analysis.assert_valid_module m;
34  print_string (Llvm.string_of_llmodule m)
```

7.8 SVG rendering - draw.c

```
1  //-------------------------------------------------------------
2  //  ssol/draw.c
3  //   This is the main C file for our SVG file rendering
4  //   utility as well as complex type constructors,
5  //      to be linked in with the LLVM code
6  //   generated by the SSOL compiler.
7  //
8  //   This code is mostly not ours! The original
9  //   source code was taken from here:
10 //    http://www.code-in-c.com/writing-svg-library-c/
11 //
12 //   See also svg.h and svg.c
13 //
14 //      Additional authorship: Daniel Mesko
15 //
16 //-------------------------------------------------------------
17
18 #include<stdio.h>
19 #include<math.h>
20 #include<time.h>
21
22 #include "svg.h"
23
24 #define POINT 0
25
26 //-------------------------------------------------------------
27 // FUNCTION PROTOTYPES
28 //-------------------------------------------------------------
29 void read_canvas(struct canvas_node *node, svg *psvg);
30 int draw(struct canvas canv, char *filename);
31
32 // ------------------------------------------------------------
33 // TYPE CONSTRUCTORS
34 // ------------------------------------------------------------
35
36 struct point Point(double x, double y)
37 {
38   struct point pt;
39   pt.x = x;
40   pt.y = y;
41   return pt;
42 }
43
44 struct curve Curve(struct point ep1, struct point ep2, struct point cp1, struct point
       cp2)
45 {
46   struct curve cv;
47   cv.ep1 = ep1;
48   cv.ep2 = ep2;
49   cv.cp1 = cp1;
50   cv.cp2 = cp2;
51   return cv;
52 }
53
54 struct canvas Canvas(double x, double y)
55 {
56   struct canvas c;
57
58   c.x = x;
59   c.y = y;
60   c.first = 0;
61   return c;
62 }
63
64 // ------------------------------------------------------------
65 //  CANVAS READING/SVG RENDERING FUNCTIONS
66 // ------------------------------------------------------------
67
68 int draw(struct canvas canv, char *filename)
```

```
69  {
70    svg* psvg;
71    psvg = svg_create(canv.x, canv.y);
72
73    if (psvg == NULL) {
74      fprintf(stderr, "could not store SVG meta data, malloc returned null");
75      exit(1);
76    }
77    else{
78     read_canvas(canv.first, psvg);
79
80     svg_finalize(psvg);
81     svg_save(psvg, filename);
82     svg_free(psvg);
83    }
84
85    return 0;
86  }
87  void read_canvas(struct canvas_node *node, svg *psvg)
88  {
89    // Walk the canvas node list, render each curve element
90
91    if (node == NULL)
92      return;
93
94    struct canvas_node *next = node->next;
95    struct curve *ct = node->ct;
96
97    int ep1x = (int) ct->ep1.x;
98    int ep1y = (int) ct->ep1.y;
99    int ep2x = (int) ct->ep2.x;
100   int ep2y = (int) ct->ep2.y;
101   int cp1x = (int) ct->cp1.x;
102   int cp1y = (int) ct->cp1.y;
103   int cp2x = (int) ct->cp2.x;
104   int cp2y = (int) ct->cp2.y;
105
106   svg_bezier(psvg, ep1x, ep1y, ep2x, ep2y, cp1x, cp1y, cp2x, cp2y);
107
108   read_canvas(next, psvg);
109
110 }
```

7.9 SVG rendering - svg.c

```
1  #include<stdlib.h>
2  #include<stdbool.h>
3  #include<stdio.h>
4  #include<string.h>
5  #include<math.h>
6
7  #include"svg.h"
8
9
10 // ------------------------------------------------------------
11 // STATIC FUNCTION appendstringtosvg
12 // ------------------------------------------------------------
13 static void appendstringtosvg(svg* psvg, char* text)
14 {
15     int l = strlen(psvg->svg) + strlen(text) + 1;
16
17     char* p = realloc(psvg->svg, l);
18
19     if(p)
20     {
21         psvg->svg = p;
22     }
23
24     strcat(psvg->svg, text);
25 }
26
```

```
27  // ------------------------------------------------------------
28  // STATIC FUNCTION appendnumbertosvg
29  // ------------------------------------------------------------
30  static void appendnumbertosvg(svg* psvg, int n)
31  {
32      char sn[16];
33
34      sprintf(sn, "%d", n);
35
36      appendstringtosvg(psvg, sn);
37  }
38
39  // ------------------------------------------------------------
40  // FUNCTION svg_create
41  // ------------------------------------------------------------
42  svg* svg_create(int width, int height)
43  {
44      svg* psvg = malloc(sizeof(svg));
45
46    char* style = "<style type='text/css'><![CDATA[\n.Curve { "
47     "fill:none; stroke:black; stroke-width:5 }\n]]></style>\n";
48
49      if(psvg != NULL)
50      {
51          *psvg = (svg){.svg = NULL, .width = width, .height = height, .finalized =
                false};
52
53          psvg->svg = malloc(1);
54
55          sprintf(psvg->svg, "%s", "\0");
56
57          appendstringtosvg(psvg, "<svg width='");
58          appendnumbertosvg(psvg, width);
59          appendstringtosvg(psvg, "px' height='");
60          appendnumbertosvg(psvg, height);
61          appendstringtosvg(psvg, "px' xmlns='http://www.w3.org/2000/svg' version='1.1'
                xmlns:xlink='http://www.w3.org/1999/xlink'>\n");
62      appendstringtosvg(psvg, style);
63
64          return psvg;
65      }
66      else
67      {
68          return NULL;
69      }
70  }
71
72  // ------------------------------------------------------------
73  // FUNCTION svg_finalize
74  // ------------------------------------------------------------
75  void svg_finalize(svg* psvg)
76  {
77      appendstringtosvg(psvg, "</svg>");
78
79      psvg->finalized = true;
80  }
81
82  //------------------------------------------------------------------
83  // FUNCTION svg_bezier
84  //------------------------------------------------------------------
85  void svg_bezier(svg *psvg, int x1, int y1, int x2, int y2, int cx1, int cy1,
86    int cx2, int cy2)
87  {
88    char path[500];
89    snprintf(path, 500, "<path class='Curve' d='M%d,%d C%d,%d %d,%d %d,%d' />",
90     x1,y1,cx1,cy1,cx2,cy2,x2,y2);
91     //x1, y1, x2, y2, cx1, cy1, cx2, cy2);
92
93    appendstringtosvg(psvg, path);
94  }
95
96
```

```
97  //-------------------------
98  // FUNCTION svg_text
99  //-------------------------
100 void svg_text(svg* psvg, int x, int y, char* fontfamily, int fontsize, char* fill,
        char* stroke, char* text)
101 {
102   char buf[200];
103
104   int ret = sprintf(buf, "<text x='50' y='50' style='font-size:50px'>%s</text>", text)
        ;
105   appendstringtosvg(psvg, buf);
106 }
107
108 //-------------------------------------------------------------
109 // FUNCTION svg-line
110 //-------------------------------------------------------------
111 void svg_line(svg* psvg, char* stroke, int strokewidth, int x1, int y1, int x2, int y2
        )
112 {
113     appendstringtosvg(psvg, "    <line stroke='");
114     appendstringtosvg(psvg, stroke);
115     appendstringtosvg(psvg, "' stroke-width='");
116     appendnumbertosvg(psvg, strokewidth);
117     appendstringtosvg(psvg, "px' y2='");
118     appendnumbertosvg(psvg, y2);
119     appendstringtosvg(psvg, "' x2='");
120     appendnumbertosvg(psvg, x2);
121     appendstringtosvg(psvg, "' y1='");
122     appendnumbertosvg(psvg, y1);
123     appendstringtosvg(psvg, "' x1='");
124     appendnumbertosvg(psvg, x1);
125     appendstringtosvg(psvg, "' />\n");
126 }
127
128 // -------------------------------------------------------------
129 // FUNCTION svg_save
130 // -------------------------------------------------------------
131 void svg_save(svg* psvg, char* filepath)
132 {
133     if(!psvg->finalized)
134     {
135         svg_finalize(psvg);
136     }
137
138     FILE* fp;
139
140     fp = fopen(filepath, "w");
141
142     if(fp != NULL)
143     {
144         fwrite(psvg->svg, 1, strlen(psvg->svg), fp);
145
146         fclose(fp);
147     }
148 }
149
150 //-------------------------------------------------------------------
151 // FUNCTION svg_free
152 //-------------------------------------------------------------------
153 void svg_free(svg* psvg)
154 {
155     free(psvg->svg);
156
157     free(psvg);
158 }
```

7.10 SVG rendering - svg.h

```
1 #include<stdlib.h>
2 #include<stdbool.h>
3 #include<stdio.h>
```

```
4  #include<math.h>
5
6  // ------------------------------------------------------------
7  // STRUCT definitions
8  // ------------------------------------------------------------
9  typedef struct svg
10 {
11     char* svg;
12     int height;
13     int width;
14     bool finalized;
15 } svg;
16
17
18 struct point
19 {
20   double x;
21   double y;
22 };
23
24
25 struct curve
26 {
27   struct point ep1;
28   struct point ep2;
29   struct point cp1;
30   struct point cp2;
31 };
32
33
34 struct canvas_node
35 {
36   struct canvas_node *next;
37   struct curve *ct;
38 };
39
40 struct canvas
41 {
42   float x;
43   float y;
44   struct canvas_node *first;
45 };
46
47 // ------------------------------------------------------------
48 // FUNCTION PROTOTYPES
49 // ------------------------------------------------------------
50 svg* svg_create(int width, int height);
51 void svg_finalize(svg* psvg);
52 void svg_print(svg* psvg);
53 void svg_save(svg* psvg, char* filepath);
54 void svg_free(svg* psvg);
55 void svg_bezier(svg *psvg, int x1, int y1, int x2, int y2, int cx1, int cy1,
56   int cx2, int cy2);
57 void svg_line(svg* psvg, char* stroke, int strokewidth, int x1, int y1, int x2, int y2
      );
58 void svg_fill(svg* psvg, char* fill);
59 void svg_text(svg* psvg, int x, int y, char* fontfamily, int fontsize, char* fill,
      char* stroke, char* text);
```

7.11 Test Suite

Test Program:

```
1  int main(){
2    int i[3];
3    i = {0, 1, 2, 3};
4    return 0;
5  }
```

Expected Output:

```
1  Fatal error: exception Failure("illegal assignment int[3] = int[4] in i = [ 0, 1, 2,
       3,] for identifier i")
```

Test Program:

```
int main(){
  char c = {'a','b',2};
  return 0;
}
```

Expected Output:

```
Fatal error: exception Failure("array elements are not of same type")
```

Test Program:

```
int main()
{
    int arr = {};
    return 0;
}
```

Expected Output:

```
Fatal error: exception Failure("cannot have array literal with 0 elements")
```

Test Program:

```
int main()
{
    int x = {1,2,3,4};
    float y = 3.5;

    return x[y];
}
```

Expected Output:

```
Fatal error: exception Failure("expected Int for array index value, but was given (
    float : y)")
```

Test Program:

```
int main()
{
    char x = 't';
    char t;
    t = t[5];
    return 0;
}
```

Expected Output:

```
Fatal error: exception Failure("cannot access index (int : 5) of t: it has type char")
```

Test Program:

```
int main(){
  int can = Canvas(100.0, 100.0);
  return 0;
}
```

Expected Output:

```
Fatal error: exception Failure("illegal assignment")
```

Test Program:

```
int main() {
  Canvas("100.0", "100.0");
  return 0;
}
```

Expected Output:

```
Fatal error: exception Failure("illegal argument found String expected float in
    100.0")
```

Test Program:

```
1  int main(){
2    int c = Curve(Point(10.0, 10.0), Point(10.0, 10.0), Point(10.0, 10.0), Point(10.0,
         10.0));
3    return 0;
4  }
```

Expected Output:

```
1  Fatal error: exception Failure("illegal assignment")
```

Test Program:

```
1  int main(){
2    Curve(30, 30);
3    return 0;
4  }
```

Expected Output:

```
1  Fatal error: exception Failure("expecting 4 arguments in Curve(30, 30)")
```

Test Program:

```
1  int main(){
2    Curve(Point(1.0, 1.0), Point(2.0, 2.0), Point(3.0, 3.0), 4.0);
3    return 0;
4  }
```

Expected Output:

```
1  Fatal error: exception Failure("illegal argument found float expected Point in 4.0")
```

Test Program:

```
1  int main()
2  {
3      Point p = Point(3.2,5.5);
4      p.x = "hiya";
5  }
```

Expected Output:

```
1  Fatal error: exception Failure("illegal member access -  expression type is not a
       field")
```

Test Program:

```
1  int main()
2  {
3      Point p = Point(5.0,3.0);
4      p.2+1;
5      return 0;
6  }
```

Expected Output:

```
1  Fatal error: exception Failure("invalid field usage")
```

Test Program:

```
1  int main()
2  {
3      float f = 5.3;
4      f.x;
5      return 0;
6  }
```

Expected Output:

```
1  Fatal error: exception Failure("type float does not have members")
```

Test Program:

```
1  int main()
2  {
3    -3.5 && 1; /* Float with AND? */
4    return 0;
5  }
```

Expected Output:

```
Fatal error: exception Failure("illegal binary operator float && int in -3.5 && 1")
```

Test Program:

```
int main()
{
  -3.5 && 2.5; /* Float with AND? */
  return 0;
}
```

Expected Output:

```
Fatal error: exception Failure("illegal binary operator float && float in -3.5 &&
    2.5")
```

Test Program:

```
int main() {
 Point p = Point(1.0,1.0);
 Canvas can = Canvas(100.0,100.0);
 can |= p;
 return 0;
}
```

Expected Output:

```
Fatal error: exception Failure("illegal binary operator Canvas |= Point in can |= p")
```

Test Program:

```
int main(){
  Curve pt = Point(20.0, 50.0);
  return 0;
}
```

Expected Output:

```
Fatal error: exception Failure("illegal assignment")
```

Test Program:

```
int main() {
  Point(1, 2);
  return 0;
}
```

Expected Output:

```
Fatal error: exception Failure("illegal argument found int expected float in 1")
```

Test Program:

```
int main() {
  int i = {0, 1, 2, 3};
  i[0] = 5;
  print(i[0]);

  int x = 10;
  i[0] = x;
  int z = i[0];
  print(z);
  return 0;
}
```

Expected Output:

```
5
10
```

Test Program:

```
int main(){
  int i = {1, 2, 3};
  int a = 5;
  a = a + i[0];
  print(a);
  return 0;
}
```

Expected Output:

```
1 6
```

Test Program:

```
1  int main(){
2    int x =  {0, 1, 2, 3, 4};
3    char y[5];
4    y = {'a', 'b', 'c', 'd', 'e'};
5    return 0;
6  }
```

Expected Output: (nothing)

Test Program:

```
1  int main() {
2    Canvas can;
3    can.x = 5.0;
4   can.y = 6.0;
5   printf(can.x);
6   printf(can.y);
7    return 0;
8  }
```

Expected Output:
(nothing)

Test Program:

```
1  int main() {
2    Canvas(5.0, 6.0);
3    return 0;
4  }
```

Expected Output: (nothing)

Test Program:

```
1  int main(){
2    Curve c = Curve(Point(10.0, 10.0), Point(10.0, 10.0), Point(10.0, 10.0), Point(10.0,
         10.0));
3    float cx = c.cp1.x;
4    printf(cx);
5    return 0;
6  }
```

Expected Output:

```
1 10
```

Test Program:

```
1  int main() {
2    Curve c = Curve(Point(1.0, 1.0), Point(1.0, 1.0), Point(1.0, 1.0), Point(1.0, 1.0));
3    printf(c.ep1.x);
4   return 0;
5  }
```

Expected Output:

```
1 1
```

Test Program:

```
1  int main()
2  {
3    Point pt1 = Point(23.0,376.0);
4    Point pt2 = Point(420.0,69.0);
5    Point pt3 = Point(241.0,379.0);
6    Point pt4 = Point(495.0, 174.0);
7
8
9    Curve cv1 = Curve(pt1, pt2, pt3, pt4);
10   Curve cv2 = Curve(Point(132.4,151.2), pt1, Point(921.0, 941.3), pt2);
11   Curve cv3 = Curve(pt2, Point(436.3,421.1), pt2, pt1);
```

```
12
13    Canvas can = Canvas(1000.0, 1000.0);
14
15    can |= cv1;
16    can |= cv2;
17    can |= cv3;
18
19    draw(can, "hiya.svg");
20    return 0;
21  }
```

Expected Output:
(nothing)

Test Program:

```
1  int fib(int x)
2  {
3    if (x < 2) return 1;
4    return fib(x-1) + fib(x-2);
5  }
6
7  int main()
8  {
9    print(fib(0));
10   print(fib(1));
11   print(fib(2));
12   print(fib(3));
13   print(fib(4));
14   print(fib(5));
15   return 0;
16 }
```

Expected Output:

```
1  1
2  1
3  2
4  3
5  5
6  8
```

Test Program:

```
1  int main(){
2    Point a = Point(10.0, 10.0);
3    Point b = Point(10.0, 10.0);
4    Point c = Point(10.0, 10.0);
5    Point d = Point(10.0, 10.0);
6
7    float fx = a.x;
8    printf(fx);
9
10   Curve cur = Curve(a, b, c, d);
11   float sum = cur.cp1.x - 5.0;
12
13   printf(sum);
14
15   return 0;
16 }
```

Expected Output:

```
1  10
2  5
```

Test Program:

```
1  int main() {
2    Curve c = Curve(Point(1.0, 1.0), Point(1.0, 1.0), Point(1.0, 1.0), Point(1.0, 1.0));
3   Canvas can = Canvas(100.0,100.0);
4   can |= c;
5   return 0;
6  }
```

Expected Output:
(nothing)

Test Program:

```
int main(){
  String str = "hello world";
  prints(str);
  return 0;
}
```

Expected Output:

```
hello world
```

Test Program:

```
int main()
{
 Point x;
 float y = 5.2;
 x = Point(y, 3.4);
 printf(x.x);
 return 0;
}
```

Expected Output:

```
5.2
```

Test Program:

```
int main() {
  Point(1.0, 1.0);
  return 0;
}
```

Expected Output:
(nothing)