SSOL

*Simple Shape Oriented Language*

Madeleine Tipp
mrt2148
Manager / Test Architect

Jeewan Farias
jtf2126
Language Designer

Daniel Mesko
dpm2153
System Architect

December 19, 2018
# Contents

Introduction .......................................................... 3

1 Language Tutorial .................................................. 4
   1.1 Requirements .................................................. 4
   1.2 Compiler ....................................................... 4
      1.2.1 Building the Compiler .................................. 4
      1.2.2 Running the Compiler .................................. 4
      1.2.3 Output files .............................................. 4
   1.3 Code Walkthrough ............................................. 4
      1.3.1 Full Sample Source ..................................... 4
      1.3.2 Annotated Code Fragments ............................. 5

2 Language Reference Manual ....................................... 6
   2.1 Lexical Conventions ........................................... 6
      2.1.1 Identifiers ............................................... 6
      2.1.2 Keywords ............................................... 6
      2.1.3 Literals .................................................. 7
      2.1.4 Comments ............................................... 7
      2.1.5 Punctuators .............................................. 7
      2.1.6 Whitespace .............................................. 7
   2.2 Types ................................................................ 7
      2.2.1 Primitives ............................................... 7
      2.2.2 Complex Types .......................................... 7
      2.2.3 Arrays ................................................... 8
   2.3 Syntax ................................................................ 8
      2.3.1 Type Specifiers .......................................... 8
      2.3.2 Arrays ................................................... 9
      2.3.3 Operators ............................................... 9
      2.3.4 Statements .............................................. 10
      2.3.5 Functions ............................................... 10
   2.4 Execution ....................................................... 11
      2.4.1 Scope ..................................................... 11
      2.4.2 main() .................................................... 11
   2.5 Built-In Functions .............................................. 11
      2.5.1 draw() .................................................... 11
      2.5.2 printf() ................................................... 11
      2.5.3 print() .................................................... 11
      2.5.4 printbig() ............................................... 12
      2.5.5 prints() ................................................... 12
   2.6 Sample Code ..................................................... 12

3 Project Plan .......................................................... 13
   3.1 Development Process ......................................... 13
      3.1.1 Planning .................................................. 13
      3.1.2 Specification and Development ....................... 13
      3.1.3 Testing ................................................... 13
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

<table>
<thead>
<tr>
<th>package</th>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td>opam</td>
<td>1.2.2</td>
</tr>
<tr>
<td>llvm</td>
<td>6.0.0</td>
</tr>
<tr>
<td>ocamlfind</td>
<td>1.8.0</td>
</tr>
</tbody>
</table>

1.2 Compiler

1.2.1 Building the Compiler

Run `make` to build the compiler executable, `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 file format.

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.

```c
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;

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

Finally, draw() the canvas, which will translate all the curves that have been added to their SVG repre-
sentations, 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 or 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

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td>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]*)</td>
</tr>
<tr>
<td>Float</td>
<td>A sequence of digits separated by a '.' representing an un-named float-point number (i.e. [0-9]<em>.[0-9][0-9]</em>)</td>
</tr>
<tr>
<td>Char</td>
<td>A single character enclosed by single quotation marks representing an un-named character. (i.e. `.' )</td>
</tr>
<tr>
<td>String</td>
<td>A sequence of characters enclosed by a pair of double quotation marks representing an un-named string. (i.e. <code>&quot;.*&quot; </code>)</td>
</tr>
<tr>
<td>Bool</td>
<td>8-bit boolean variable, either true or false</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>4 byte signed integer</td>
</tr>
<tr>
<td>float</td>
<td>8 byte floating-point decimal number</td>
</tr>
<tr>
<td>bool</td>
<td>1 byte Boolean value</td>
</tr>
<tr>
<td>char</td>
<td>1 byte ASCII character</td>
</tr>
<tr>
<td>String</td>
<td>array of ASCII characters</td>
</tr>
</tbody>
</table>

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:

\[
\text{Point}(\text{float } x, \text{float } y)
\]

Curve

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

\[
\text{Curve}(\text{Point } \text{ep1}, \text{Point } \text{ep2}, \text{Point } \text{cp1}, \text{Point } \text{cp2})
\]

The constructor creates a Curve defined by endpoints \( a \) and \( b \) and control points \( c_1 \) and \( c_2 \). 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 \| = \text{ operator}. Canvas objects are outputted to files via the \text{draw} library function.

A Canvas object is instantiated using the following constructor:

\[
\text{Canvas}(\text{float } x, \text{float } y)
\]

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 separate or combined statements.

Primitives

\[
\begin{align*}
\text{int } x; \\
x &= 3; \\
\text{String } \text{myString} &= \text{"hello"}; \\
\text{bool } b &= \text{true};
\end{align*}
\]

Complex Types

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

\[
\begin{align*}
&\text{//Canvas can be instantiated with default size or with a user specified size} \\
&\text{Canvas } \text{can2} = \text{Canvas}(100.0,100.0); \\
&\text{Point } \text{pt} = \text{Point}(10.0,20.0); \\

&\text{/*/ Create a straight line by declaring a Curve} \\
&\text{* using the end points as the control points} \\
&\text{*/}
\end{align*}
\]

8
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.

```java
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.

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

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

2.3.5 Functions

Declaration

Functions are declared as follows:
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>)`;

---

11
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

```c
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 individual's 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

<table>
<thead>
<tr>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>September 16</td>
<td>Creation of code repository, sample program and initial commit</td>
</tr>
<tr>
<td>September 19</td>
<td>Submitted Project Proposal</td>
</tr>
<tr>
<td>October 15</td>
<td>Submitted Scanner, Parser, and Language Reference Manual</td>
</tr>
<tr>
<td>November 14</td>
<td>Submitted &quot;Hello World&quot; program</td>
</tr>
<tr>
<td>December 1</td>
<td>Completed Semantic Checking</td>
</tr>
<tr>
<td>December 10</td>
<td>Completed Code Generation</td>
</tr>
<tr>
<td>December 18</td>
<td>Presented project and Fibonacci spiral program</td>
</tr>
<tr>
<td>December 19</td>
<td>Submitted report and code</td>
</tr>
</tbody>
</table>

3.4 Roles and Responsibilities

<table>
<thead>
<tr>
<th>Team Member</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Madeleine Tipp</td>
<td>Manager and Test Architect</td>
</tr>
<tr>
<td>Jeevan Farias</td>
<td>Language Designer</td>
</tr>
<tr>
<td>Daniel Mesko</td>
<td>System Architect</td>
</tr>
</tbody>
</table>

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/Oopam 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 (`scanner.ml`)

• **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

```plaintext
commit f05d8ebc49a9544cc8558769b5729dcf28adefe
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Wed Dec 19 23:49:02 2018 -0500

Added tests to tarball dependency

commit a137f47460cfd19363cd4ce6a3092d9252b8429
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Wed Dec 19 22:53:15 2018 -0500

Added array and field fail tests

commit fb20513e52786d450d4e59a969af31e7a839c0f
Author: Daniel Mesko <dpmesko@gmail.com>

  cleaned up indentation

commit c0dbadf4c18bd2cfc72106ff9848df83c164c63
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Wed Dec 19 22:04:00 2018 -0500

  spacing in semant, more array error tests

commit 8b298b3b87d688694703e14d1b1ce26df6b385
Author: mtipp <mrt2148@barnard.edu>
Date: Wed Dec 19 18:37:46 2018 -0500

  Update field tests and delete /temptests.
```
Update array tests. fails oob.

Updated testall, added pipend test

Testfailures (#39)
* updated testall, added pipend test

Testfailures (#38)
* test updates. still failures on makefile printbig.o
* Update tests and err messages.
* fixed testall script
* all tests pass

added a printf function to print strings (#37)

Merge pull request #36 from dpmesko/cleanup

Replaced tabs with spaces in OCaml code, cleaned up indentation, removed old comments – generally BEAUTIFIED
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 569a809419666d87e9673576187a0ce700ca226d
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Tue Dec 18 16:36:33 2018 -0500
temporarily commented out testall target from Makefile

commit 50f2c6899e93c37213611a329445c39dd59d4803
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Tue Dec 18 16:35:09 2018 -0500
removed unnecessary comment

commit d7134a4d0bf6d6b0ae366b5907273f35f697de
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 1abef0f505068f10f9152236c939cb3a847edcf
Merge: c209552 a864d9b
Author: Daniel Mesko <dpmesko@gmail.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
Removed unnecessary files from compiler directory

commit c16ebca7cac43db3682fa74b47685d702ae8e85a
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Mon Dec 17 21:26:56 2018 -0500
Converted Constructor/SConstructor to Call/SCall so that we can check args to constructors in semant

commit c209552d6c056046955d9956b4ab52c5d6dcf79
Author: jvnft <jeevanfarias@gmail.com>
Date: Mon Dec 17 20:37:11 2018 -0500
Revert "augmented print call to print anything. identifiers print as 'Identifier: %name' (#30)"

This reverts commit 58b6b73ae0226649dce41b81cf2f39e6e6d89a1.

commit 58b6b73ae0226649dce41b81cf2f39e6e6d89a1
Author: jvnft <jeevanfarias@gmail.com>
Date: Mon Dec 17 20:23:40 2018 -0500
augmented print call to print anything. identifiers print as 'Identifier: %name'

commit df8139a36f4620a06d4cd31d4bef17c258691759
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Mon Dec 17 16:38:37 2018 -0500
removed pipe operator from all parts of compiler – no longer a feature of SSOL

commit 100e41ed71ed5035f1cd782c6c514c6fd7030b4c
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Mon Dec 17 03:12:11 2018 -0500
modified test-draw

commit 080b5368e12c9febd4e5ff1ce0bdfb15bcb49999
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Mon Dec 17 02:56:04 2018 -0500
Removed unnecessary arg to Canvas constructor, but still returns garbage from linked C code. Cleaned dup Makefile target structure

commit 0f4fbee06bcbe29826acc7ba3be05340a954772db
Author: jvnft <jeevanfarias@gmail.com>
Date: Mon Dec 17 01:21:59 2018 -0500
Control flow tests (#29)

* got rid of some unused variable warnings
* added codegen: expr->SArrayAssign
* added charlit
* added user defined function calls
* reversed arg checking in codegen -> SCall
* matched some patterns to get rid of warnings
* Change test extensions and testall.sh from .mc to .ssol.
* updated ssol script to not remove files if u accidentally run with the file extension
* updated tests and fixed output files

commit dfce6af19ee0fa457742af173a539dbd7477244f
Author: Daniel Mesko <dpmesko@gmail.com>
some changes to canvas constructor, draw()...trying to fix double/float issue

commit 32bb8490554465ac5c3d8e6e2e2f1b6b031f3831
Author: jvntf <jeevanfarias@gmail.com>
Date: Mon Dec 17 00:46:39 2018 −0500

Function returns (#28)
Function return types

commit cf988518c8542c70ad301e7e652bc8e505a779
Author: jvntf <jeevanfarias@gmail.com>
Date: Sun Dec 16 23:57:37 2018 −0500

Pipend dan (#26)

* removing files that should have never been committed
* 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 7e1bc613d9b956c6b32839309047566e42d975b
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
Merge pull request #24 from dpmesko/draw-link

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

Fixed error with Canvas construction – L const_pointer_null is garbage

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

SVG linking (attempted)
[WIP] Trying to figure out Canvas struct problems

commit 433e8bd91fbd0a39d68d6135adaf82cbe47c531e
Author: jvntf <jeevanfarias@gmail.com>
Date: Sun Dec 16 15:57:09 2018 -0500

Codegen expr (#20)
* got rid of some unused variable warnings
* added codegen:expr->SArrayAssign
* added charlit
* added user defined function calls
* reversed arg checking in codegen -> SCall
* matched some patterns to get rid of warnings
* field going in the right direction, not working yet
* codegen SField working and compiling, assignment will throw semantic error when doing field assignment to variable
* got field codegene working, but problems with assignment to a variable
* changed order of arguments for Canvas constructor to match the others
* field assignment working...semantic errors get thrown at runtime instead of compile time

commit a083461f0794514eb455de2d68e65eal1a6583be9
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Sun Dec 16 15:37:49 2018 -0500

[WIP] fixing draw linking

commit 7159bc2181ae5c4782f85102eddee4e9738ac733
Merge: 53a26b322f1be3d
Author: mtipp <mrt2148@barnard.edu>
Date: Sun Dec 16 05:09:20 2018 -0500

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

commit 53a26b3f6ed7ef6d1c616efa999f3e7a905fa5fa
Author: mtipp <mrt2148@barnard.edu>
Date: Sun Dec 16 05:08:58 2018 -0500

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

commit 841c3d17b800002bf0d9d3b6b619c13ad6d7678a
Author: mtipp <mrt2148@barnard.edu>
Date: Sun Dec 16 05:00:34 2018 -0500

Start work on pipend.

commit 82952eabf3e7a8d6bf21fc8400ae7f73158327f
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Sun Dec 16 04:02:42 2018 -0500

Updated SVG code to draw bezier cues, walk the canvas_nodes, added test main function and Makefile target

commit 830f1b1bf0d9d8b16178b074434990eabbc9a5d59
Merge: 35289c42583f1f
Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
Date: Sun Dec 16 01:31:00 2018 -0500

Merge pull request #19 from dpmesko/svg-draw

semant, codegen, SVG lib updated for new definition of draw(), codegen updated for
new canvas_node definition

commit 2583f1f1dd5b35296911b4eaea3eadd186efd9b
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Sat Dec 15 23:22:14 2018 -0500

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

commit 23fbe3d0a7ac664058322e355d9cd5b4ddc806a8
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Sat Dec 15 22:56:36 2018 -0500

removing files that should have never been commited

commit e237876fd6b60ce68d544cd96327a769af4a58ef
Merge: e5733a8 c0f1cd2
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Sat Dec 15 22:47:36 2018 -0500

merging in codegen−maddy, fixing conflicts

commit e5733a8285e274f447d7414c2d15bc978ac56585
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Sat Dec 15 22:43:17 2018 -0500

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

commit 22ca73d5aa8247b6f7ad68ceada0d0de02b9a79c
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Sat Dec 15 22:36:30 2018 -0500

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

commit c0f1cd297ce863557dfe2e4310e51c8837d5821ca
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Sat Dec 15 17:38:35 2018 -0500

Adding/modifying tests for complex type constructor usage and assignment

commit a7e0e0d6de15449576e4f41cda3e573df454f
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Sat Dec 15 17:35:47 2018 -0500

cleaning up unnecessary files

commit f9de2aa23fa4fbb45943cafee8543e8837db52d
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Sat Dec 15 17:35:18 2018 -0500

[WIP] Trying to fix identifiers not being valid arguments to type constructors

commit d9fc1599f3de66d825a81c8a6d59f3b18a696c7
Merge: 5dfaae3 35289cd
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Sat Dec 15 15:14:12 2018 -0500

getting new symbol table, Field access from semant

Merge branch ‘master’ into codegen−maddy

commit 35289cd2154a4529cab9e5a1491a76c9f94f1c3c
Merge: f7a81b ec7df3a
Author: Daniel Mesko <35789221@dpmesko@users.noreply.github.com>
Date: Sat Dec 15 15:12:52 2018 -0500

Merge pull request #18 from dpmesko/semant−dan
Reverted to old symbol table format, created member_map_of_type function.

commit 5dffaee38b3ea67d6a4005a0f73d876b5f9713fa0
Author: mtipp <mrt2148@barnard.edu>
Date: Sat Dec 15 14:59:18 2018 -0500

spacing changes.

commit 14e1ba41a0a681c510f181865675c1dd7d4810c
Merge: 6db26a6 fe7a81b
Author: mtipp <mrt2148@barnard.edu>
Date: Fri Dec 14 21:54:30 2018 -0500

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

commit fe7a81b84d6b3a5e9f3a1b1f7c453aef3f4b61f2
Author: mtipp <mrt2148@barnard.edu>
Date: Fri Dec 14 21:50:00 2018 -0500

Canvas test passes with int expressions.

commit ec7df3add0cb1d173419c79b8fd71c3b7303574b
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Fri Dec 14 12:16:58 2018 -0500

Reverted to old symbol table format, created member_map_of_type function, implemented Field expr check, removed some unused variables, commented out code

commit 6db26a607bc21826ebd6b404f47240809b3e794
Author: mtipp <mrt2148@barnard.edu>
Date: Thu Dec 13 18:57:36 2018 -0500

Canvas constructor compiles.

commit 75c0cc6122c7748be9841271ec983126fb2e6360
Merge: cf00a91 dca0005
Author: mtipp <mrt2148@barnard.edu>
Date: Thu Dec 13 16:09:12 2018 -0500


commit cf00a91a59a53489381a02581e7f0681d0dd9eb0
Author: mtipp <mrt2148@barnard.edu>
Date: Thu Dec 13 16:06:48 2018 -0500

Start work on canvas constructor.

commit dca00055fa9d0e78ed7ed462f2a2e8252e060bd2
Merge: 1937a11 96c0c73
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Wed Dec 12 19:02:13 2018 -0500

Merge pull request #17 from dpmesko/semant-dan

Finished array assignment, array literals, array access

commit 96c0c7336b8818a56488a034334d56c5a99906f7b
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Wed Dec 12 18:57:58 2018 -0500

Fixed check_assign for arrays to also check sizes. Added an appropriate testcase to array.ssol

commit d54b35ddbb88e3c8c8b719427fc5d81d0076543c
Merge: 5165c8f 19e555c
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Wed Dec 12 18:48:50 2018 -0500

Merging in recent array semant/codegen progress, fixing conflicts
commit 19e555c0841e57da9696bcf3edd1065e9ba160b
Author: Jeevan <jeevanfarias@gmail.com>
Date:   Wed Dec 12 18:40:04 2018 -0500

debugged array access, vdeclassign, arraylit

commit 5165c8f835f8444db3657ad94598ba653b9b4f81a
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 52f8c90f430f949efed62637bd0fa3a98dc9024f6
Merge: b420194 1937a11
Author: mtipp <mrt2148@barnard.edu>
Date:   Wed Dec 12 18:38:59 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 b716e3af4d0e2c5d78bbd1a265f8d8b6c964a4d
Author: mtipp <mrt2148@barnard.edu>
Date:   Wed Dec 12 17:06:49 2018 -0500

Start defining canvasonde_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 a5871b4ce2605f105610c6c718a94fc0e6d375e6
Author: Daniel Mesko <dpmesko@gmail.com>
Date:   Wed Dec 12 16:19:40 2018 -0500

Fixed arraylit checking

commit 80845b670b079d6fa7c49c8415488781f997258b
Merge: 60fd57 1937a11
Author: Jeevan <jeevanfarias@gmail.com>
Date:   Wed Dec 12 16:16:07 2018 -0500

merging array lit changes from master

commit 60fd57b1566aa308b6f2aac48fd6c19c7f4e64
Author: Jeevan <jeevanfarias@gmail.com>
Date:   Wed Dec 12 16:14:10 2018 -0500

started codegen expr arraylit

commit 1937a110a359a268a9d240506ea905031d210b83
Merge: 6019a00 05dded
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
Merging master, fixing conflicts, deleting backups/assembly files

Commented out broken symbol map code to get it to compile

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 constructors.
* fixed ADecl and ADecl ast type

Point assignment passes test pt−assign.ssol

Curve only passes test with point constructors.

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

 Added semantic check for Mod

Updated creation of initial symbol table to have (type, option StringMap) as values, with member variable maps for complex types
Semantic checking and codegen for expressions and statements

added failed tests.

messing with initial StringMap

Uncommented test script target in Makefile

Implemented pipe, pipend, arraylit patterns; began converting symbol map to name
-> (ty, option StringMap), passing as arg to expr, member map access function

Removed lambda map build nonsense, got it to compile

added new param to expr calls

Last merge to bring me up to date

Merging other branches before I do any more duplicate work...

merged in semant–jf
commit 6d8baf3fad0256a64673a5a47bbaae5828a53728
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Thu Dec 6 16:30:53 2018 −0500
[WIP] modifying symbol table to store (typ, Stringmap) pairs for members of complex types; passing current symbol map to expr

commit 49e3c93306c32bb3645cd784d8aef459296f76db
Author: Jeevan <jeevanfarias@gmail.com>
Date: Wed Dec 5 21:48:36 2018 −0500
var decl and assignment working
commit d94eb8c1ee7f89faa7d6575ca8820511518
Author: Jeevan <jeevanfarias@gmail.com>
Date: Wed Dec 5 20:47:00 2018 −0500
modified semantics and codegen to allow for var decl. works but value is wrong
commit fb548a50fc02a81646d2ff7f9f02a14861bd
Author: Jeevan <jeevanfarias@gmail.com>
Date: Wed Dec 5 19:48:20 2018 −0500
added local var map to codegen−>expr, need to make semantic mods to make work
commit 8868393260051249d07ae692f7e5f02f194861bd
Author: Jeevan <jeevanfarias@gmail.com>
Date: Wed Dec 5 18:18:24 2018 −0500
commented out ADecl
commit e144d08c69d8eee28f44d9ff66d9787ee398436e
Author: Jeevan <jeevanfarias@gmail.com>
Date: Tue Dec 4 21:22:11 2018 −0500
added VDeclAssign to codegen. NEED TO CHANGE expr to take local string map
commit edd053dc9851db4187fd58699c6612882cd821ef
Author: Jeevan <jeevanfarias@gmail.com>
Date: Tue Dec 4 12:04:23 2018 −0500
added a local var map to codegen.stmt and added SVDecl pattern
commit 6843fb6b5a1a4e1f01d8a7b0c6e0a5bc79be6bfa
Author: Jeevan <jeevanfarias@gmail.com>
Date: Wed Dec 5 19:36:02 2018 −0500
fixed semantic:: stmt → SBlock
commit bf0976be959d559434f070d9430e176fb4ddfd4d5
Author: mtipp <mrt2148@barnard.edu>
Date: Tue Dec 4 15:41:12 2018 −0500
SConstructor for point compiles
commit 17b8b9f8f7f5777886a0d08fc4dd76c4b6ad4a89
Author: Jeevan <jeevanfarias@gmail.com>
Date: Mon Dec 3 16:08:53 2018 −0500
matched pattern on [], need to verify that this works
commit d95d8edef181fc1473213ca757e099182f47c785
Author: Jeevan <jeevanfarias@gmail.com>
Date: Mon Dec 3 16:07:29 2018 −0500
fixed partial function appl
added block_locals to stmt

commit c7c44c381438d56cfdddbde720f063184eaca127a
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Mon Dec 3 00:13:17 2018 -0500

complex type constructor calls in semant, pretty printing in ast, sast

commit cb898bc2ef2d2ad506d866bd8712d064f2f3c7494d
Author: Jeevan <jeevanfarias@gmail.com>
Date: Sat Dec 1 1 7:34:48 2018 -0500

added VDdecl, VDeclAssign, ADecl to semant

commit 57f3eb6ada797b2d5597beb5d56b34f3b5d734
Author: jvntf <jeevanfarias@gmail.com>
Date: Wed Nov 28 13:43:36 2018 -0500

changes from hello world (#10)

* wrote readme, ready for submission
* removed locals and vdecls_list from fdecl
* removed refs to locals from ast, sast, semant
* edit removal of locals
* fixed type problems in parser, still need to handle alloc in codegen
* added new zip file
* removed zip file from repo

commit 7e38cbce012137b53fece9a47ccaf06e429e16ba
Author: Daniel Mesko <35789221-dpmesko@users.noreply.github.com>
Date: Mon Nov 19 01:50:14 2018 -0500

Update README.md

commit 185362785e7d6aecf716c9d6d7063e95c0cb2134
Author: Daniel Mesko <35789221-dpmesko@users.noreply.github.com>
Date: Mon Nov 19 01:49:38 2018 -0500

Update README.md

commit b3bdc72d5cf3439aa0002ba1f9ed31f6465f7557
Author: Jeevan <jeevanfarias@gmail.com>
Date: Mon Nov 19 01:36:16 2018 -0500

added new zip file

commit 92ad5e0cc5014a0508027097e2633ae8911842da
Author: Jeevan <jeevanfarias@gmail.com>
Date: Mon Nov 19 01:23:20 2018 -0500

fixed type problems in parser, still need to handle alloc in codegen

commit 77c44a312a3be67aacc6fe49170125c69b1abf74
Author: Jeevan <jeevanfarias@gmail.com>
Date: Mon Nov 19 01:12:15 2018 -0500

edit removal of locals

commit 93241efa99ae558cced68a1a53386de30a3e4e2
Author: Jeevan <jeevanfarias@gmail.com>
Date: Mon Nov 19 00:50:01 2018 -0500

removed refs to locals from ast, sast, semant

commit 382c20fbe7c798597cc7d914318f2ae0344a8f24
Author: Jeevan <jeevanfarias@gmail.com>
Date: Mon Nov 19 00:49:30 2018 -0500
removed locals and vdecls_list from fdecl

commit 589b0dffa8f45d91037ec25768101dadd2e7db4b
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

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 338d8a477f08bf177c2e9526146dbee33561f96
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

commit cb13cac3b5ca7a3979f4d173e0c1f125991e6325
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

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

commit d599802ea2b39954e09c1fd1b1648c2a001ac22
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 3b4cf8860d3e4e20bb78434ca6035e41d15732b  
Author: Jeevan <jeevanfarias@gmail.com>  
Date: Tue Nov 13 12:15:10 2018 -0500  
Hello world

commit bd8e9efa44e5cf2dfe2ca069db7a2fbb8c7f1e  
Author: Jeevan <jeevanfarias@gmail.com>  
Date: Tue Nov 13 11:42:17 2018 -0500  
Wrote comp script, baby test program

commit 3ba54a9d339e37fd3d5ccf89f6b031adc46d9ac7  
Author: Daniel Mesko <dpmesko@gmail.com>  
Date: Mon Nov 12 21:13:33 2018 -0500  
Moved draw C files

commit 821bfaa58395a54a588fd47529be46da5746be  
Merge: 2f8a8f413d480fb  
Author: Daniel Mesko <dpmesko@gmail.com>  
Date: Mon Nov 12 20:57:36 2018 -0500  
I WANT NEW STUFF

commit 2f8a8f44ae3e84e33a5af3894a9ae2cf1a1aaf8d  
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 df6838bd1520af4cd5b6490730a176c0a819a8a  
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 bda93762951546e6ec0c8c34c0af1e9f7f2f88ca7c  
Author: Daniel Mesko <dpmesko@gmail.com>  
Date: Mon Nov 12 19:34:01 2018 -0500  
Added expr rule for complex type constructors

commit 13d80f8c8ce9c05735c73592f4c7bd0ac00d4645f  
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 4bf55301510952017737254daed4760b9e73df510  
Author: Daniel Mesko <dpmesko@gmail.com>  
Date: Sun Nov 11 16:49:14 2018 -0500  
Fixed spacing

commit 522aa609c8364e752f28ce51337d90890d31097b  
Merge: 8eb4e2b e736316
fixed merge conflicts with Maddy's branch

commit e736316a88fb563e0db6a4d5b456f414fe601a7
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 16:17:38 2018 -0500

commit e736316a88fb563e0db6a4d5b456f414fe601a7
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 e736316a88fb563e0db6a4d5b456f414fe601a7
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

[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 shiop.

commit 8eb4e2b93631562e3a5aaaf32afbf18d8bb0b2926
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Sun Nov 11 15:07:42 2018 -0500

add mod shiop.

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 70dfeaa8a84366b9de16b210ab3823b5912537d
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Sun Nov 11 14:00:19 2018 -0500

added ArrayLit logic

commit 99ec9be514bb81c015e05e68b642668f6bda3a1c
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Sun Oct 28 19:46:39 2018 -0400

Added TODOs

commit 2e9d6b58e5808a795318f5ace097461c4cd12ee9
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 7b6564a424b7ff19b08b57c2a6d4f344ce26c037
Merge: cd6324f 76fb8ad
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Fri Oct 26 16:27:10 2018 -0400

Merge pull request #7 from dpmesko/new-microc

commit 76fb8ad7ebc8e776e2744a00a3c7b3d5dc1b2bfc
Author: Daniel Mesko <dpmesko@gmail.com>
Merged SSOL scanner/parser/ast with newest MicroC source

commit cd6324f65e8ec5703fc50c696683c5d3028ba0a7
Merge: 7b3230d b088278e
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 b088278e6995b1e827b689b090c2ee367ca2943
Author: Jeevan <jeevanfarias@gmail.com>
Date: Mon Oct 15 19:35:46 2018 -0400

updated lrm

commit e3a43185d28767905d572d101d4aed3ec0ece6c0
Merge: af8626be e2d409ff
Author: Jeevan <jeevanfarias@gmail.com>
Date: Mon Oct 15 19:15:10 2018 -0400

added array access

commit e2d409ff1c171ad36ff5dd20546d813c2aa7d677
Author: Jeevan <jeevanfarias@gmail.com>
Date: Mon Oct 15 19:15:10 2018 -0400

added array access

commit 9f164636e47a4b0360916bdfed4d3628da7a5cf
Author: Jeevan <jeevanfarias@gmail.com>
Date: Mon Oct 15 18:41:50 2018 -0400

added newest version of lrm

commit 382c57950166f5a536ff56a7d7b6524bd9f1d32f2
Author: Jeevan <jeevanfarias@gmail.com>
Date: Mon Oct 15 18:39:26 2018 -0400

fixed assoc

commit f1cfca030786e9c6b2263ed771603d1f3929a4
Author: Jeevan <jeevanfarias@gmail.com>
Date: Mon Oct 15 18:34:16 2018 -0400

added object fields to parser and ast

commit 275ce285df4296d083ae1e4e6288377e39f9c
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Mon Oct 15 18:11:45 2018 -0400

Updated Makefile dependencies, added array literals to ast

commit 15d43e254595e2937687273cd30c3beedca9f5d8
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
Added char literals, changed file names in Makefile, renamed stuff....

commit b2823c384eb6ea68c3180090683a227694bcc61b
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Sat Oct 13 18:42:59 2018 -0400

[WIP] Added (some) operator types to ast

commit a3c47ac1c03c86344ef90b23e474ecbe82fc605c
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Sat Oct 13 18:42:07 2018 -0400

Added tokens to parser, updated typ rule, added MOD to expr rule

commit cf845b6a65a80c15eb04b136eed8a067b08f3a80
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Sat Oct 13 18:41:13 2018 -0400

Added missing }, renamed LITERAL token to INT\_LITERAL

commit 7b3230de3c511c55ebe184babf7187e30ccd9ee4
Merge: 43212185fd811bb
Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
Date: Fri Oct 12 16:07:07 2018 -0400

Merge pull request #4 from dpmesko/scanner

added (probably) all tokens to scanner

commit 4321218c0be09af0488de631fafee00c5feb7011
Author: Jeevan <jeevanfarias@gmail.com>
Date: Fri Oct 12 16:03:29 2018 -0400

added lrm pdf

commit 95be444d93414c4a06eff772d5d1555b29f97ad2
Merge: 334b93b8176066
Author: jvntf <jeevanfarias@gmail.com>
Date: Fri Oct 12 15:59:31 2018 -0400

Merge pull request #5 from dpmesko/lrm

added lrm latex source

commit 81760663314d887c24df260be36ea1a57125a0ec
Author: Jeevan <jeevanfarias@gmail.com>
Date: Fri Oct 12 15:59:08 2018 -0400

added lrm latex source

commit 5fd81bb8d8a2444138447c6851c8c94daf03ed0c
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Fri Oct 12 14:37:29 2018 -0400

Made it so that float literals can omit the integer part, added DOT token

commit c487df90263b87dab885399d6d8540d820edf2e4
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Wed Oct 10 21:03:15 2018 -0400

Added more elements, single line comments

commit cfd403b78493eab2824d2ea3874140181cd30c35
Author: Daniel Mesko <dpmesko@gmail.com>
Date: Wed Oct 10 20:39:52 2018 -0400

Began adding token rules to scanner

commit 334b93bc40921a5d9ac24e7a70df3854a06b33eb
Merge: 91c88a308b7d22
Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
Date: Fri Oct 5 19:52:52 2018 -0400
Merge pull request #3 from dpmesko/proposal

Proposal merge

commit 91c88a348f0ebf7d0c0ab68f28e52da3b2e922c
Merge: 5431dd05 5116d87
Author: Daniel Mesko <35789221@dpmesko@gmail.com>
Date: Fri Oct 5 19:48:44 2018 -0400

Merge pull request #2 from dpmesko/setup

Added MicroC compiler source, renamed microc.ml, now ssol.ml

commit 5116d87d184ba96004a0780bba47abb6c365634d
Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
Date: Fri Oct 5 19:45:28 2018 -0400

Added MicroC compiler source, renamed microc.ml, now ssol.ml

commit 5431dd502d3f6747db76d3a106b34d8ea66b6a49
Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
Date: Wed Oct 3 01:02:38 2018 -0400

Update README.md

commit 08b7d225e0b0519d1c06f4c8241fbf7a934e28408
Author: Jeevan <jeevanfarias@gmail.com>
Date: Wed Sep 19 19:45:21 2018 -0400

capitalized

commit 32b89a241676189bebafe3b81f2be8ac49a6e474
Author: Jeevan <jeevanfarias@gmail.com>
Date: Wed Sep 19 19:32:25 2018 -0400

yadl sample program

commit d371c7fc267c5e8a7a9d7d35b8b55ef00f2dec83
Merge: 163c583 4874d18
Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
Date: Sun Sep 16 18:02:36 2018 -0400

Merge pull request #1 from dpmesko/proposal

Sample Program for Proposal

commit 4874d18f8b224dcb2176d467d4a1371c8fb8fc
Author: Jeevan <jeevanfarias@gmail.com>
Date: Sun Sep 16 17:37:20 2018 -0400

added for loop

commit 163c583e00ea9fcb3520bb55ab23fa4de1415a4c
Author: Jeevan <jeevanfarias@gmail.com>
Date: Sun Sep 16 17:29:33 2018 -0400

started sample program

commit cd934b1874bc4ccbc5b5710342bca1bc1b9db36d
Author: Daniel Mesko <35789221+dpmesko@users.noreply.github.com>
Date: Sun Sep 16 17:14:21 2018 -0400

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 \texttt{\texttt{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:

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

The LLVM output from test-curveconstruct.ssol:

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

%canvasnode = type { %canvasnode*, { { double, 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, 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 } } %c,
      return 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(3.0, 3.0), 4.0);
    return 0;
  }

  fail-curveconstruct-typeargs.err:

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

  The LLVM output from fail-curveconstruct-typeargs.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 $1 | sed 's/.ssol$//''

    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
    }
}
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 getopt 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

41
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 development 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 (* Ocamlex 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 }
7.2 Parser

/* Ocamlyacc parser for SSOL */
/* Authors: Jeevan Farias, Madeleine Tipp, Daniel Mesko */

open Ast

%{

%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA
%token LBRACK RBRACK
%token PLUS MINUS TIMES MOD DIVIDE ASSIGN
%token NOT EQ NEQ LT LEQ GT GEQ AND OR
%token DOT PIPE PIPEND
%token RETURN IF /*ELIF*/ ELSE FOR WHILE INT BOOL FLOAT VOID
%token BREAK CONTINUE
%token CHAR STRING POINT CURVE CANVAS
%token <int> LITERAL
%token <bool> BLIT
%token <char> CHAR_LITERAL
%token <string> STRING_LITERAL
%token <id> ID FLIT
%token EOF

%start program
%type <Ast.program> program

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left OR
%left AND
%left NOT
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE
%left MOD
%left PIPE PIPEND
%left DOT
%right NOT NEQ
program:
  decls EOF { $1 }

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

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

formals_opt:
/* nothing */ { [] }
| formal_list { $1 }

formal_list:
typ ID {
  | formal_list COMMA typ ID { ($3,$4) :: $1 }
}
typ:
  INT { Int }
| BOOL { Bool }
| FLOAT { Float }
| VOID { Void }
| CHAR { Char }
| STRING { String }
| POINT { Point }
| CURVE { Curve }
| CANVAS { Canvas }

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

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

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

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

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

expr:
LITERAL { Literal($1) }
| FLIT { Fliteral($1) }
| BLIT { BoolLit($1) }
| ID { Id($1) }
| CHAR_LITERAL { CharLit($1) }
STRING_LITERAL { StringLit($1) }

| expr PLUS expr { Binop($1, Add, $3) }
| expr MINUS expr { Binop($1, Sub, $3) }
| expr TIMES expr { Binop($1, Mult, $3) }
| expr DIVIDE expr { Binop($1, Div, $3) }
| expr MOD expr { Binop($1, Mod, $3) }
| expr EQ expr { Binop($1, Equal, $3) }
| expr NEQ expr { Binop($1, Neq, $3) }
| expr LT expr { Binop($1, Less, $3) }
| expr LEQ expr { Binop($1, Leq, $3) }
| expr GT expr { Binop($1, Greater, $3) }
| expr GEQ expr { Binop($1, Geq, $3) }
| expr AND expr { Binop($1, And, $3) }
| expr OR expr { Binop($1, Or, $3) }
| 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 | And | Or | Pipend  
5
6 type uop = Neg | Not  
7
8 type typ = Int | Bool | Float | Void | Char | String | Point | Curve | Canvas  
9 | Array of typ * int  
10
11 type bind = typ * string  
12
13 type expr =  
14 | Literal of int  
15 | Fliteral of string  
16 | BoolLit of bool  
17 | CharLit of char  
18 | StringLit of string  
19 | ArrayLit of expr list  
20 | Id of string  
21 | Binop of expr * op * expr  
22 | Field of string * expr  
23 | Unop of uop * expr  
24 | Assign of string * expr  
25 | Access of string * expr  
26 | ArrayAssign of string * expr * expr  
27 | Call of string * expr list  
28 | Noexpr  
29
30 type stmt =  
31 | VDecl of typ * string  
32
type func_decl = {
  typ : typ;
  fname : string;
  formals : bind list;
  body : stmt list;
}

type program = bind list * func_decl list

(* Pretty-printing functions *)

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

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

let rec string_of_typ = function
  Int -> "int"
  Bool -> "bool"
  Float -> "float"
  Void -> "bool"
  Char -> "char"
  String -> "String"
  Point -> "Point"
  Curve -> "Curve"
  Canvas -> "Canvas"
  Array ( t , n) -> (string_of_typ t) ^ "[ " ^ (string_of_int n) ^ " ]"

let rec string_of_expr = function
  Literal(l) -> string_of_int l
  Fliteral(l) -> l
  BoolLit(true) -> "true"
  BoolLit(false) -> "false"
  CharLit(l) -> String.make 1 l
  StringLit(l) -> l
  ArrayLit (arr) -> 
    "[ " ^ (List.fold_left (fun lst elem -> lst ^ " " ^ string_of_expr elem ^ " " ^ string_of_expr elem ^ " "]") ^ "arr") ^ ""]"
  Field(s, f) -> s ^ ". " ^ string_of_expr f
  Id(s) -> s
  Binop(e1, o, e2) -> 
    string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
  Unop(o, e) -> string_of_uop o ^ string_of_expr e
  Assign(v, e) -> v ^ "=" ^ string_of_expr e
  Access(a, i) -> a ^ "[ " ^ string_of_expr i ^ " ]")
  ArrayAssign(arr, index, rval) -> arr ^ "[ " ^ string_of_expr index ^ " ]") ^ 
    string_of_expr rval
  Call(f, el) -> 
    f ^ "(" ^ String.concat ", " ^ (List.map string_of_expr el) ^ ")")
let rec string_of_vdecl (t, id) = string_of_typ t ^ ^ id ^ ^ "\n"

let rec string_of_fdecl fdecl =
  string_of_typ fdecl ^ ^ "
  fdecl.fname ^ ^ "String.concat",
  (List.map snd fdecl.formals)^ ^ "\n"

(* String.concat " (List.map string_of_vdecl fdecl.locals) ^ *)

String.concat " (List.map string_of_stmt fdecl.body) ^ " ^ "\n"

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

String.concat "\n" (List.map string_of_fdecl funcs)

7.4 Semantic Checker

(* Semantic checking for the SSOL compiler *)
(* Authors: Jeevan Farias, Madeleine Tipp, Daniel Mesko *)

open Ast
open Sast

module StringMap = Map.Make (String)

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

Check each global variable, then check each function *)

let check (globals, functions) =

(* Verify a list of bindings has no void types or duplicate names *)

let check_binds (kind : string) (binds : bind list) =
  List.iter (function
    | Void, b -> raise (Failure ("illegal void " ^ kind ^ ^ " " ^ b))
    | _ -> ()
  ) binds

let rec dupes = function
  | [] -> ()
  | [(.,n1) :: (_,n2) :: _,] when n1 = n2 ->
    raise (Failure ("duplicate " ^ kind ^ ^ " " ^ n1))
  | _ :: t -> dupes t

in
dups (List.sort (fun (.,a) (_,b) -> compare a b) binds)

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

check_binds "global" globals;

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

(* Collect function declarations for built-in functions: no bodies *)
let built_in_decls =
let add_bind map (name, retyp, formlist) = StringMap.add name {
  ty = retyp;
  fname = name;
  formals = formlist;
(*) locals = []; *) body = [] } map
in List.fold_left add_bind StringMap.empty [ ("print", Void, [(Int, "x")]) ;
("printb", Void, [(Bool, "x")]);
("printf", Void, [(Float, "x")]);
("printbig", Void, [(Int, "x")]);
("prints", Void, [(String, "x")]);
("draw", Void, [(Canvas, "can"); (String, "filename")]);
("Point", Point, [(Float, "x"); (Float, "y")]);
("Curve", Curve, [(Point, "ep1"); (Point, "ep2")];
(Point, "cp1"); (Point, "cp2")]);
("Canvas", Canvas, [(Float, "x"); (Float, "y")]);}

in

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

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

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

in

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

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

(* Raise an exception if the given rvalue type cannot be assigned to
 the given lvalue type *)
let check_assign lvalue rvalue err =
  match lvalue with
  Array(lt, ls) ->
    (match rvalue with
      Array(rt, rs) -> if (lt == rt && ls == rs) then lvalue else raise (Failure err)
    | _ -> raise (Failure err))
  | _ -> if lvalue == rvalue then lvalue else raise (Failure err)

in

(* Build initial symbol table with globals and formals *)
let globmap = List.fold_left (fun m (ty, name) -> StringMap.add name ty m)
  StringMap.empty (globals @ func.formals)

in

(* Return type of a symbol from supplied symbol table *)
let type_of_identifier locals s =
  try StringMap.find s locals
  with Not_found -> raise (Failure ("unddeclared identifier " ^ s))
(* Return member symbol map for a particular type *)
let member_map_of_type ty = match ty with
  | Point -> List.fold_left (fun m (ty, name) -> StringMap.add name ty m) StringMap.empty
  | Curve -> List.fold_left (fun m (ty, name) -> StringMap.add name ty m) StringMap.empty
  | _ -> raise (Failure ("type " ^ string_of_typ ty ^ " does not have members"))

in

(* Return a semantically-checked expression, i.e., with a type *)

let rec expr locals = function
  | Literal l -> (Int, SLiteral l)
  | Fliteral l -> (Float, SFliteral l)
  | BoolLit l -> (Bool, SBoolLit l)
  | CharLit l -> (Char, SCharLit l)
  | StringLit l -> (String, SStringLit l)
  | ArrayLit e llist ->
    let typmatch t (ty,_) =
      if t == ty then
        ty
      else
        raise (Failure ("array elements are not of same type"))
    and slist = List.map (fun e -> expr locals e) llist in
    (match slist with
      | [] -> raise (Failure "cannot have array literal with 0 elements")
      | _ ->
        let ty = List.fold_left typmatch (fst (List.hd slist)) slist in
        (Array(ty, List.length slist), SArrayLit(slist))
  | Noexpr -> (Void, SNoexpr)
  | Id s -> (type_of_identifier locals s, SId s)
  | Assign(var, e) as ex ->
    let lt = type_of_identifier locals var
    and (rt, e') = expr locals e in
    (match arrtyp with
      | Array(t,_) ->
        let err = "illegal assignment " ^ string_of_typ lt ^ " = " ^ string_of_expr ex ^ " for identifier " ^ var
        in (check_assign lt rt err, SAssign(var, (rt, e')))
      | _ -> raise (Failure (err))
    )
  | Access(arr, ind) ->
    let arrtyp = type_of_identifier locals arr
    and (ityp, _) as ind' = expr locals ind in
    (match arrtyp with
      | Array(t,_) ->
        let err = "illegal assignment " ^ string_of_typ arrtyp ^ " = " ^ string_of_expr (fst ex') in
        (match arrtyp with
          | Array(t,_) ->
            let err = "illegal assignment " ^ string_of_typ arrtyp ^ " = " ^ string_of_expr (fst ex') in
            (match arrtyp with
              | Array(t,_) ->
                let err = "illegal assignment " ^ string_of_typ arrtyp ^ " = " ^ string_of_expr (fst ex') in
                (match arrtyp with
                  | Array(t,_) -> (check_assign t (fst ex') err, SArrayAssign(arr, ind', ex'))
                  | _ -> raise (Failure (err))
                )
              | _ -> raise (Failure (err))
            )
          | _ -> raise (Failure (err))
        )
      | _ -> raise (Failure (err))
    )
  | Field(obj, mem) ->
    let ty = type_of_identifier locals obj
    let memmap = member_map_of_type ty in
    let smem = match mem with
      | Assign(v,e) as ex ->
        let ty = type_of_identifier memmap v in
        (match e with
          | Fliteral _ ->
            let lt = StringMap.find v memmap
            and (rt, e') = expr locals e in

let err = "illegal assignment of object field" \
string_of_typ lt " = " \
string_of_typ rt " in " \
string_of_expr ex " for identifier Field." v \
in (check_assign lt rt err, SAssign(v, (rt, e')))
| Id s -> (ty, SAssign(v, (ty, SId s)) )
| _ -> raise (Failure ("illegal member access - \
" expression type is not a field")))

(fst smem, SField(obj, smem))

| Unop(op, e) as ex ->
let (t, e') = expr locals e in
let t = match op with
Neg when t = Int | t = Float -> t
| Not when t = Bool -> Bool
| _ -> raise (Failure ("illegal unary operator " \
string_of_uop op " string_of_typ t " \
in " string_of_expr ex))

| Binop(e1, op, e2) as e ->
let (t1, e1') = expr locals e1
and (t2, e2') = expr locals e2 in
(* All binary operators require operands of the same type *)
let same = t1 = t2 in
(* Determine expression type based on operator and operand types *)
let ty = match op with
Add | Sub | Mult | Div when same && t1 = Int -> Int
| Add | Sub | Mult | Div when same && t1 = Float -> Float
| Equal | Neq when same -> Bool
| Less | Leq | Greater | Geq when same && (t1 = Int | t1 = Float) -> Bool
| And | Or when same && t1 = Bool -> Bool
| Mod when same && t1 = Int -> Int
| Pipend when t1 = Canvas && t2 = Curve -> Canvas
| _ -> raise (Failure ("illegal binary operator " \
string_of_typ t1 " " string_of_op op " " \
string_of_typ t2 " in " string_of_expr e))

| Call(fname, args) as call ->
let fd = find_func fname in
let param_length = List.length fd.formals in
if List.length args != param_length then
raise (Failure ("expecting " string_of_int param_length " \
arguments in " string_of_expr call))
else let check_call (ft _, _) e =
let (et, e') = expr locals e in
let err = "illegal argument found " string_of_typ et " \
"expected " string_of_typ ft " in " string_of_expr e
in (check_assign ft et err, e')
in
let args' = List.map2 check_call fd.formals args
in (fd.ty, SCall((fname, args')))
in
let check_bool_expr locals e =
let (t', e') = expr locals e
and err = "expected Boolean expression in " string_of_expr e
in if t' != Bool then raise (Failure err) else (t', e')
in
(* Return a semantically-checked statement i.e. containing sexprs *)

let rec check_stmt locals = function
Block sl as
let rec check_block block_locals ssl= function
[Return _ as s] -> ssl @ [check_stmt block_locals s]
| Return (::: _) -> raise (Failure "nothing may follow a return")
| Block sl :: ss -> [check_stmt block_locals (Block sl)]

@ (check_block block_locals ssl ss)
match s with
  | Void -> raise (Failure ("illegal void local " ^ name))
  | _ -> let block_locals = StringMap.add name t block_locals
        in [check_stmt block_locals s] @ check_block block_locals
        ssl ss
  | VDeclAssign (t, name, e) ->
    if t == Void then raise (Failure ("illegal void local " ^ name))
    else
      let sx = expr block_locals e in
      let typ = (match f (fst sx) with
                | Array (tp, s) -> if tp == t then Array (tp, s)
                | _ -> if fst (sx) == t then fst (sx)
                | _ -> raise (Failure ("Array literal is of inconsistent type")))
      in
        let block_locals = StringMap.add name typ block_locals in
        [check_stmt block_locals s] @ check_block block_locals ssl ss
  | ADecl (t, name, n) ->
    if t == Void then raise (Failure ("illegal void local " ^ name))
    else
      let block_locals = StringMap.add name (Array (t, n)) block_locals
      in
        let sx = expr block_locals e in
        let typ = type_of_identifier locals s in
        SDeclAssign (ty, sx)
        in
          let block_locals = StringMap.add name (Array (t, n)) block_locals
          in
            let block_locals = StringMap.add name (Array (t, n)) block_locals
            in
              let sx = expr block_locals e in
              let typ = type_of_identifier locals s in
              SDeclAssign (ty, sx)
              in
                raise (Failure ("illegal assignment"))
  | Expr e -> SExpr (expr block_locals e)
  | If (p, b1, b2) -> SIf (check_bool_expr locals p, check_stmt locals b1, check_stmt locals b2)
  | For (e1, e2, e3, st) ->
    SFor (expr locals el, check_bool_expr locals e2, expr locals e3, check_stmt locals st)
  | While (p, s) -> SWhile (check_bool_expr locals p, check_stmt locals s)
  | Return e -> let (t, e') = expr locals e in
    if t = func.typ then SReturn (t, e')
    else raise (Failure ("return gives " ^ string_of_typ t " expected " ^
                     string_of_typ func.typ " in " ^ string_of_expr e))
    in (* body of check_function *)
    { styp = func.typ;
      sfname = func.fname;
      sformals = func.formals;
      sbody = match check_stmt globmap (Block func.body) with
             | SBlock (sl) -> sl
             | _ -> raise (Failure ("internal error: block didn't become a block?
             | ))
    }
        in (globals, List.map check_function functions)

7.5 SAST

(* Semantically-checked Abstract Syntax Tree and functions for printing it *)
(* Authors: Jeevan Farias, Madeleine Tipp, Daniel Mesko *)

open Ast

type sexpr = typ * sx
and sx =
  SLiteral of int
| SLiteral of string
| SBoolLit of bool
| SCharLit of char
| SStringLit of string
| SArrayLit of sexpr list
type sstmt =
  SVDecl of typ * string
| SADeclAssign of typ * string * sexpr
| SADecl of typ * string * int
| SBlock of sstmt list
| SExpr of sexpr
| SReturn of sexpr
| SIf of sexpr * sstmt * sstmt
| SFor of sexpr * sexpr * sexpr * sstmt
| SWhile of sexpr * sstmt

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

type sprogram = bind list * sfunc_decl list

(* Pretty-printing functions *)

let rec string_of_sexpr (t, e) =
  "(" string_of_typ t " " (match e with
  SLiteral(l) -> string_of_int l
| SLiteral(l) -> l
| SBoolLit(true) -> "true"
| SBoolLit(false) -> "false"
| SCharLit(l) -> String.make 1 l
| SArrayLit(arr) -> "]" (List.fold_left (fun lst elem -> lst " " " string_of_sexpr elem " " " ) " arr) " ]")"
| SField(t, i) -> t " . " string_of_sexpr i
| SId(l) -> l
| SBinop(o, e1, e2) -> string_of_sexpr e1 " " o " " string_of_sexpr e2
| SUnop(o, e) -> string_of_uop o " string_of_sexpr e"
| SAssign(v, e) -> v " = " string_of_sexpr e
| SArrayAssign(arr, index, rval) -> arr " [ " string_of_sexpr index " ] " string_of_sexpr rval
| SFor(s1, s2, e3, s) -> if (" string_of_sexpr e " ) " while (" string_of_sexpr e " ) " string_of_sexpr s
| SWhile(e, s) -> " while (" string_of_sexpr e " ) " string_of_sexpr s

let rec string_of_sstmt = function
  SVDecl(t, i) -> string_of_typ t " " i " \n"
| SVDeclAssign(t, i, e) -> string_of_typ t " " i " = " string_of_sexpr e " \n"
| SADecl(t, i, s) -> string_of_typ t " " i " [ " string_of_int s " ] \n"
| SBlock(stmts) -> "\n" (List.map string_of_sstmt stmts) " \n"
| SExpr(expr) -> string_of_sexpr expr " \n"
| SReturn(expr) -> " return " string_of_sexpr expr " \n"
| SIf(e, s, SBBlock([])) -> " if (" string_of_sexpr e " ) " string_of_sstmt s
| SFor(s1, s2) -> " for (" string_of_sexpr s1 " else \n" string_of_sstmt s2
| SWhile(e, s) -> " while (" string_of_sexpr e " ) " string_of_sstmt s
7.6 Code Generator

(* Semantic checking for the SSOL compiler *)
(* Authors: Jeevan Farias, Madeleine Tipp, Daniel Mesko *)

open Ast
open Sast

module StringMap = Map.Make(String)

(* Semantic checking of the AST. Returns an SAST if successful, throws an exception if something is wrong. *)
Check each global variable, then check each function *)

let check (globals, functions) =

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

(** Check global variables ***)
check_binds "global" globals;

(** Check functions ***)

(* Collect function declarations for built-in functions: no bodies *)

let built_in.decls =
let add_bind map (name, retyp, formlist) = StringMap.add name {
typ = retyp;
fname = name;
formals = formlist;
(* locals = []; *) body = []} map
in List.fold_left add_bind StringMap.empty [
("print", Void, [(Int, "x")]);
("printf", Void, [(Float, "x")]);
("printbig", Void, [(Int, "x")]);
("prints", Void, [(String, "x")]);
("draw", Void, [(Canvas, "can"); (String, "filename")]);
("Point", Point, [(Float, "x"); (Float, "y")]);
("Curve", Curve, [(Point, "ep1"); (Point, "ep2")];
(Point, "cp1"); (Point, "cp2")]);
("Canvas", Canvas, [(Float, "x"); (Float, "y")]);
in
(* Add function name to symbol table *)
let add_func map fd =
  let built_in_err = "function " ^ fd.fname ^ " may not be defined"
and dup_err = "duplicate function " ^ fd.fname
and make_err er = raise (Failure er)
and n = fd.fname (* Name of the function *)
in match fd with (* No duplicate functions or redefinitions of built−ins *)
  | _ when StringMap.mem n built_in_decls -> make_err built_in_err
  | _ when StringMap.mem n map -> make_err dup_err
  | _ -> StringMap.add n fd map

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

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

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

(* Raise an exception if the given rvalue type cannot be assigned to
the given lvalue type *)
let check_assign lvalue rvalue err =
  match lvalue with
  | Array(lt, ls) ->
    (match rvalue with
      | Array(rt, rs) -> if (lt == rt && ls == rs) then lvalue else raise (Failure err)
      | _ -> raise (Failure err))
  | _ -> if lvalue == rvalue then lvalue else raise (Failure err)
in

(* Build initial symbol table with globals and formals *)
let globmap = List.fold_left (fun m (ty, name) -> StringMap.add name ty m)
  StringMap.empty (globals @ func.formals)
in

(* Return type of a symbol from supplied symbol table *)
let type_of_identifier locals s =
  try StringMap.find s locals
  with Not_found -> raise (Failure ("undeclared identifier " ^ s))
in
let member_map_of_type ty = match ty with
  | Point | Canvas ->
    List.fold_left (fun m (ty, name) -> StringMap.add name ty m)
    StringMap.empty [(Float, "x");
  | Curve ->
    List.fold_left (fun m (ty, name) -> StringMap.add name ty m)
    StringMap.empty [(Point, "e1");
  | _ -> raise (Failure ("type " ^ string_of_type ty ^ " does not have members"))
in

let rec expr locals = function
  | Literal l -> (Int, SLiteral l)
  | Fliteral l -> (Float, SFliteral l)
  | BoolLit l -> (Bool, SBoolLit l)
  | CharLit l -> (Char, SCharLit l)
let typmatch t (ty, .) =  
  if t == ty then 
    ty 
  else 
    raise (Failure ("array elements are not of same type")) 
and slist = List.map (fun e -> expr locals e) elist in 
(match slist with 
  | [] -> raise (Failure "cannot have array literal with 0 elements") 
  | _ -> 
    let ty = List.fold_left typmatch (fst (List.hd slist)) slist in 
    (Array (ty, List.length slist), SArrayLit (slist)) ) 
| Noexpr -> (Void, SNoexpr) 
| Id s -> (type of identifier locals s, SId s) 
| Assign (var, e) as ex -> 
  let lt = type of identifier locals var 
  and (rt, e') = expr locals e in 
  let err = "illegal assignment " ^ string_of_typ lt " = " ^ 
  string_of_typ rt " in " ^ string_of_expr ex " for identifier " ^ var 
  in (check_assign lt rt err, SAssign(var, (rt, e')))) 
| Access (arr, ind) -> 
  let arrtyp = type of identifier locals arr 
  and (ITY, e') = expr locals ind in 
  (match arrtyp with 
    | Array (t, .) -> 
      let (rt, e') = expr locals e in 
      let err = "illegal assignment " ^ string_of_typ arrtyp " = " ^ 
      string_of_typ (fst ex) " for identifier " ^ arr 
      in (check_assign rt err, SArrayAssign (arr, ind, ex)) ) 
    | raise (Failure "cannot access index " ^ expr ind " of " ^ arr " : it has type " ^ string_of_typ arrtyp) ) 
| ArrayAssign (arr, ind, ex) -> 
  let arrtyp = type of identifier locals arr 
  and ind' = expr locals ind 
  and ex' = expr locals ex in 
  let err = "illegal assignment " ^ string_of_typ arrtyp " = " ^ 
  (check_assign t (fst ex') err, 
  SArrayAssign(arr, ind', ex')) in 
  (match arrtyp with 
    | Array (t, .) -> 
      raise (Failure err)) ) 
| Field (obj, mem) -> 
  let ty = type of identifier locals obj in 
  let memmap = member_map_of_type ty in 
  let smem = match mem with 
    Assign (v, e) as ex -> 
      let ty = type of identifier memmap v in 
      (match e with 
        | Fliteral (_, .) -> 
          let lt = StringMap.find v memmap 
          and (rt, e') = expr locals e in 
          let err = "illegal assignment of object field" ^ 
            string_of_typ lt " = " ^ 
            string_of_typ rt " in " ^ 
            string_of_expr ex " for identifier Field." ^ v 
          in (check_assign lt rt err, SAssign(v, (rt, e')))) 
        | Id s -> (ty, SAssign(v, (ty, SId s))) 
        | _ -> raise (Failure ("illegal member access - " ^ 
                              " expression type is not a field"))) in 
    expr memmap mem 
  in 
    (fst smem, SField(obj, smem)) 
| Unop (op, e) as ex -> 
  let (t, e') = expr locals e in 
  let ty = match op with 
    | Neg when t = Int -> t 
    | _ when t = Bool -> Bool 
    | _ -> raise (Failure ("illegal unary operator " ^ 
                              string_of_uop op " string_of_typ t " ^ 
                              " in " ^ string_of_expr ex)) 
  in (ty, SUnop (op, (t, e')))

58
Binop(e₁, op, e₂) as e ->

let (t₁, e₁') = expr locals e₁
and (t₂, e₂') = expr locals e₂ in

(* All binary operators require operands of the same type *)

let same = t₁ = t₂ in

(* Determine expression type based on operator and operand types *)

let ty = match op with
  | Add | Sub | Mult | Div when same && t₁ = Int -> Int
  | Add | Sub | Mult | Div when same && t₁ = Float -> Float
  | Equal | Neq when same -> Bool
  | Less | Leq | Greater | Geq when same && (t₁ = Int || t₁ = Float) -> Bool
  | And | Or when same && t₁ = Bool -> Bool
  | Pipend when t₁ = Canvas && t₂ = Curve -> Canvas
  | . -> raise (Failure ("illegal binary operator " string_of_typ t₁ " " string_of_op op " " string_of_typ t₂ " in " string_of_expr e))

in (ty, SBinop ((t₁, e₁'), op, (t₂, e₂')))

| Call(fname, args) as call ->
  let fd = find_func fname in
  let param_length = List.length fd.formals in
  if List.length args != param_length then
    raise (Failure ("expecting " string_of_int param_length " arguments in " string_of_expr call))
  else
    let check_call (ft, e) =
      let (et, e') = expr locals e in
      let err = "illegal argument found " string_of_typ et " expected " string_of_typ ft " in " string_of_expr e
      in (check_assign ft et err, e')
    in
    let args' = List.map2 check_call fd.formals args
    in
    (fd.typ, SCall(fname, args'))
  in

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

(* Return a semantically-checked statement i.e. containing sexprs *)

let rec check_stmt locals = function
  Block s₁ ->
    let rec check_block block locals ssl= function
      [Return _, as s] -> ssl @ [check_stmt block.locals s]
      | [Return _, _] -> raise (Failure ("nothing may follow a return")
      | Block s₁ :: ss -> [check_stmt block.locals (Block s₁)]
                  @ (check_block block.locals ssl ss)
    s :: ss ->
      (match s with
        VDecl(t, name) ->
          (match t with
            Void -> raise (Failure ("illegal void local " name))
          |
            . -> let block_locals = StringMap.add name t block_locals
            in
            in [check_stmt block_locals s] @ check_block block_locals ssl ss)
          |
            VDeclAssign (t, name, e) ->
            if t == Void then raise (Failure ("illegal void local " name))
            else
              let sx = expr block_locals e in
              let typ = (match fst(sx) with
                Array(tp, s) -> if tp == t
                                      then Array(tp, s)
                                  else raise (Failure("Array literal is of inconsistent type")))
                |
                . -> if fst(sx) == t
                     then fst(sx)
                     else raise (Failure("illegal assignment"))
              |
              block_locals = StringMap.add name typ block_locals in
            [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 in SBlock(check_block_locals [] ssl)
278 | VDecl(t, s) -> SVDecl(t, s)
279 | VDeclAssign(_, s, e) ->
280 let sx = expr_locals e in
281 let ty = type_of_identifier locals s in
282 SVDeclAssign(ty, s, sx)
283 | ADecl(t, s, n) -> SADecl(t, s, n)
284 | Expr e -> SExpr(expr_locals e)
285 | If(p, b1, b2) -> SIf(check_bool_expr_locals p, check_stmt locals b1,
286 check_stmt locals b2)
287 | For(e1, e2, e3, st) ->
288 SFor(expr_locals e1, check_bool_expr_locals e2, expr_locals e3,
289 check_stmt locals st)
290 | While(p, s) -> SWhile(check_bool_expr_locals p, check_stmt locals s)
291 | Return e -> let (t, e') = expr_locals e in
292 if t = func_t then SReturn(t, e')
293 else raise (Failure ("return gives " ^ string_of_typ t ^ " expected " ^
294 string_of_typ func_typ in " ^ string_of_expr e))
295 in (* body of check_function *)
296 
297 {styp = func_typ;
298 sfname = func_fname;
299 sfformal = 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)

7.7 SSOL - top-level

(* Top-level of the SSOL compiler: scan & parse the input,
check the resulting AST and generate an SAST from it, generate LLVM IR,
and dump the module

Authors: Jeevan Farias, Madeleine Tipp, Daniel Mesko *)

type action = Ast | Sast | LLVMIR | Compile

let () =
let action = ref Compile in
let set_action a () = action := a in
let spec_list = [
("-a", Arg.Unit (set_action Ast), "Print the AST");
("-s", Arg.Unit (set_action Sast), "Print the SAST");
("-l", Arg.Unit (set_action LLVMIR), "Print the generated LLVM IR");
("-c", Arg.Unit (set_action Compile),
"Check and print the generated LLVM IR (default)");
] in
let usage_msg = "usage: ./microc.native [-a|-s|-l|-c] [file.mc]
let channel = ref stdin in
Arg.parse spec_list (fun filename -> channel := open_in filename) usage_msg;
let lexbuf = Lexing.from_channel !channel in
let ast = Parsers.ssol.program Scanner.token lexbuf in
match !action with
| Ast -> print_string (Ast.string_of_program ast)
| _ -> let sast = Semant.checkast in
match !action with
| Ast -> ()
| Sast -> print_string (Sast.string_of_sprogram sast)
| LLVMIR -> print_string (Llvm.string_of_llvmmodule (Codegen.translate sast))
| Compile -> let m = Codegen.translate sast in
Llvm_analysis.assert_valid_module m;
print_string (Llvm.string_of_llvmmodule m)
7.8 SVG rendering - draw.c

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

void read_canvas(struct canvas_node *node, svg *psvg) {
  // Walk the canvas node list, render each curve element
  if (node == NULL) return;
  struct canvas_node *next = node->next;
  struct curve *ct = node->ct;
  int ep1x = (int) ct->ep1.x;
  int ep1y = (int) ct->ep1.y;
  int ep2x = (int) ct->ep2.x;
  int ep2y = (int) ct->ep2.y;
  int cp1x = (int) ct->cp1.x;
  int cp1y = (int) ct->cp1.y;
  int cp2x = (int) ct->cp2.x;
  int cp2y = (int) ct->cp2.y;
  svg_bezier(psvg, ep1x, ep1y, ep2x, ep2y, cp1x, cp1y, cp2x, cp2y);
  read_canvas(next, psvg);
}

7.9 SVG rendering - svg.c

#include <stdlib.h>
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include <math.h>

#include "svg.h"

// STATIC FUNCTION appendstringtosvg
// appendstringtosvg(svg* psvg, char* text)
{  
  int l = strlen(psvg->svg) + strlen(text) + 1;
  char* p = realloc(psvg->svg, l);
  if (p) {
    psvg->svg = p;
  }
  strcat(psvg->svg, text);
}
// STATIC FUNCTION appendnumbertosvg
static void appendnumbertosvg(svg* psvg, int n)
{
    char sn[16];
sprintf(sn, "%d", n);
appendstringtosvg(psvg, sn);
}

// FUNCTION svg_create
svg* svg_create(int width, int height)
{
    svg* psvg = malloc(sizeof(svg));
    char* style = "<style type='text/css'>
    .Curve {
        fill: none; stroke: black; stroke-width: 5
    }
</style>
";
    if (psvg != NULL)
    {
        *psvg = (svg){.svg = NULL, .width = width, .height = height, .finalized = false};
        psvg->svg = malloc(1);
        sprintf(psvg->svg, "%s\0", style);
    appendstringtosvg(psvg, "<svg width=");
    appendnumbertosvg(psvg, width);
    appendstringtosvg(psvg, "px height=");
    appendnumbertosvg(psvg, height);
    appendstringtosvg(psvg, "px xmlns='http://www.w3.org/2000/svg' version='1.1'
    xmlns:xlink='http://www.w3.org/1999/xlink'>\n");
    appendstringtosvg(psvg, style);
    return psvg;
    }
    else
    {
        return NULL;
    }
}

// FUNCTION svg_finalize
void svg_finalize(svg* psvg)
{
    appendstringtosvg(psvg, "</svg>");
    psvg->finalized = true;
}

// FUNCTION svg_bezier
void svg_bezier(svg *psvg, int x1, int y1, int x2, int y2, int cx1, int cy1,
    int cx2, int cy2)
{
    char path[500];
sprintf(path, "<path class='Curve' d='M%d,%d C%d,%d %d,%d %d,%d' />
", x1, y1, cx1, cy1, cx2, cy2, x2, y2);
    //x1, y1, x2, y2, cx1, cy1, cx2, cy2);
    appendstringtosvg(psvg, path);
```c
// FUNCTION svg_text
void svg_text(svg* psvg, int x, int y, char* fontfamily, int fontsize, char* fill,
    char* stroke, char* text)
{
    char buf[200];
    int ret = sprintf(buf, "<text x='50' y='50' style='font-size:50px'>%s</text>", text);
    appendstringtosvg(psvg, buf);
}

// FUNCTION svg_line
void svg_line(svg* psvg, char* stroke, int linewidth, int x1, int y1, int x2, int y2)
{
    appendstringtosvg(psvg, "<line stroke='");
    appendstringtosvg(psvg, stroke);
    appendstringtosvg(psvg, " stroke-width='");
    appendnumberintosvg(psvg, "px' y2='");
    appendnumberintosvg(psvg, "' x2='");
    appendnumberintosvg(psvg, "' y1='");
    appendnumberintosvg(psvg, "' x1='");
    appendnumberintosvg(psvg, "'/"n");
}

// FUNCTION svg_save
void svg_save(svg* psvg, char* filepath)
{
    FILE* fp;
    fp = fopen(filepath, "w");
    if(fp != NULL)
    {
        fwrite(psvg->svg, 1, strlen(psvg->svg), fp);
        fclose(fp);
    }
}

// FUNCTION svg_free
void svg_free(svg* psvg)
{
    free(psvg->svg);
    free(psvg);
}

7.10 SVG rendering - svg.h
```
#include <math.h>

// STRUCT definitions
typedef struct svg {
    char* svg;
    int height;
    int width;
    bool finalized;
} svg;

struct point {
    double x;
    double y;
};

struct curve {
    struct point ep1;
    struct point ep2;
    struct point cp1;
    struct point cp2;
};

struct canvas_node {
    struct canvas_node *next;
    struct curve *ct;
};

struct canvas {
    float x;
    float y;
    struct canvas_node *first;
};

// FUNCTION PROTOTYPES
svg* svg_create(int width, int height);
void svg_finalize(svg* psvg);
void svg_print(svg* psvg);
void svg_save(svg* psvg, char* filepath);
void svg_free(svg* psvg);
void svg_bezier(svg* psvg, int x1, int y1, int x2, int y2, int cx1, int cy1, int cx2, int cy2);
void svg_line(svg* psvg, char* stroke, int strokewidth, int x1, int y1, int x2, int y2);
void svg_fill(svg* psvg, char* fill);
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:
```c
int main()
{
    char c = {'a', 'b', 2};
    return 0;
}
```

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

Test Program:
```c
int main()
{
    int arr = {};
    return 0;
}
```

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

Test Program:
```c
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:
```c
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:
```c
int main()
{
    int can = Canvas(100.0, 100.0);
    return 0;
}
```

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

Test Program:
```c
int main()
{
    Canvas("100.0", "100.0");
    return 0;
}
```

Expected Output:
```
Fatal error: exception Failure("illegal argument found String expected float in 100.0")
```
```
int main()
{
    int c = Curve(Point(10.0, 10.0), Point(10.0, 10.0), Point(10.0, 10.0), Point(10.0, 10.0));
    return 0;
}

Fatal error: exception Failure("illegal assignment")

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

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

Test Program:
```
int main()
{
    Curve(Point(1.0, 1.0), Point(2.0, 2.0), Point(3.0, 3.0), 4.0);
    return 0;
}

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

Test Program:
```
int main()
{
    Point p = Point(3.2, 5.5);
    p.x = "hiya";
}

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

Test Program:
```
int main()
{
    Point p = Point(5.0, 3.0);
    p.2+1;
    return 0;
}

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

Test Program:
```
int main()
{
    float f = 5.3;
    f.x;
    return 0;
}

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

Test Program:
```
int main()
{
    -3.5 && 1; /* Float with AND? */
    return 0;
}
```

Expected Output:

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

Test Program:

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

Expected Output:

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

Test Program:

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

Expected Output:

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

Test Program:

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

Expected Output:

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

Test Program:

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

Expected Output:

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

Test Program:

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

Expected Output:

```
5
```

Test Program:

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

Expected Output:

```
10
```
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: 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

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 Curve cv1 = Curve(pt1, pt2, pt3, pt4);
9 Curve cv2 = Curve(Point(132.4, 151.2), pt1, Point(921.0, 941.3), pt2);
10 Curve cv3 = Curve(pt2, Point(436.3, 421.1), pt2, pt1);
12 Canvas can = Canvas(1000.0, 1000.0);
14 can |= cv1;
16 can |= cv2;
17 can |= cv3;
18 draw(can, "hiya.svg");
20 return 0;
}

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

Expected Output:
1 10
2 5

Test Program:

1 int main()
2 {
3 Curve c = Curve(Point(1.0, 1.0), Point(1.0, 1.0), Point(1.0, 1.0), Point(1.0, 1.0));
4 Canvas can = Canvas(100.0, 100.0);
5 can |= c;
6 return 0;
7 }
Test Program:

```c
1 int main() {
2   String str = "hello world";
3   prints(str);
4   return 0;
5 }
```

Expected Output:

1 hello world

Test Program:

```c
1 int main()
2 {
3   Point x;
4   float y = 5.2;
5   x = Point(y, 3.4);
6   printf(x.x);
7   return 0;
8 }
```

Expected Output:

1 5.2

Test Program:

```c
1 int main() {
2   Point(1.0, 1.0);
3   return 0;
4 }
```

Expected Output:

(nothing)