better call **SOL**

SHAPE ORIENTED LANGUAGE FINAL REPORT

Aditya Narayanamoorthy	`an2753`	*Language Guru*
Erik Dyer	`ead2174`	*System Architect*
Gergana Alteva	`gla2112`	*Program Manager*
Kunal Baweja	`kb2896`	*Tester*

December 20, 2017

Contents

Chapter 1

Introduction

SOL (Shape Oriented Language), is a domain specific programming language that allows programmers to create 2D animations with ease, through an object-oriented approach. Engineers, programmers, scientists and designers, through SOL, have the ability to define and create objects, known as Shapes, and dictate their appearance and movements on the screen. SOLś simplicity saves developers the trouble of learning complicated third-party animation tools, without sacrificing control over behavior of objects. It compiles into LLVM IR bytecode, making it adaptable across different architectures. The produced LLVM IR bytecode can be translated further into assembly code and linked statically against a predefined library before compiling down into an executable for a specific architecture using the LLVM compiler and GCC compilers respectively. The predefined library used by SOL for graphic rendering has been built on top of SDL2, that abstracts away the lower level details for drawing and animating objects.

1.1 Background

SOL takes its inspiration from the ease of programming in *object-oriented paradigm* and the complexity of existing libraries/solutions for rendering graphics. It attempts to combine both of them and provide a language which allows developers to organize graphics into a collection of *Shapes* (much like objects) which can be easily defined, created and interacted with to produce powerful images and/or animations at a fast paced development.

SOL is commonly used to model various types of scientific data, but it can also be applicable in other domains, such as:

1. Drawing engineering models
2. Data visualization
3. Bored college students making funny memes
4. Entertaining animations

Chapter 2

Language Tutorial

This chapter provides a brief description of basic components of the language to guide programmers towards creating their first SOL! It has been divided into three parts:

1. **Environment setup** - guide to setting up the SOL compiler and development environment
2. **Language Quick Tour** - a bried descriptioin of basic components of language.
3. **Sample Program** - Create dancing line bars

2.1 Environment setup

SOL compiler has been developed and tested on Ubuntu 15.04 and Ubuntu 16.04 environments, and is capable of supporting others as well. It can work on multiple architectures as the LLVM IR bytecode can further be compiled into architecture specific executables. The environment setup tutorial assumes you have the latest version of SOL compile source code downloaded from our repository.

2.1.1 Building SOL compiler (Ubuntu 15.04 or later version)

1. SOL compiler is written in OCaml. Download the source code from our github repository:
   `https://github.com/bawejakunal/sol`
2. Download and run the bash script below, which installs the latest compatible version of OCaml compiler and opam for *-nix OS environments.
   `https://raw.githubusercontent.com/ocaml/ocaml-ci-scripts/master/.travis-ocaml.sh`
3. Configure the opam environment for importing Ocaml packages
   ```
   eval `opam config env`
   ```
4. Install LLVM compiler toolchain and ocaml bindings, for the SOL compiler
   ```
   ./install-llvm.sh
   ```
5. Download and Install SDL2 and SDL2_gfx libraries which are used by our predefined static library for graphics rendering.
   ```
   wget http://www.ferzkopp.net/Software/SDL2_gfx/SDL2_gfx-1.0.3.tar.gz
   ./install-sdl-gfx.sh
   ```

6. Build the SOL compiler as `sol.native` and the static graphic libarary as `predefined.o`
   `make all`

2.2 SOL Quick Tour

This section provides a quick tour on the data types, data structures and the shape oriented programming paradigm provided by SOL, with examples. Towards the end we walk through the steps to generate an executable SOL program. For a detailed explanation of language syntax, semantics and built-in functions refer to chapter 3.

2.2.1 Primitive data types

All variables of primitive data types are declared starting with their type followed by an identification. SOL supports the following primitives:

1. `int` - integers
2. `float` - double precision floating point number
3. `char` - single character from `ASCII` character set
4. `string` - a sequence of one or more `ASCII` characters

Example:

```
func main() {
    int x;  /* declare primitives */
    float y;
    char c;
    string s;

    x = 5;  /* assign values to primitives */
    y = 6.5;
    c = 'a';
    s = "better call SOL";
}
```

2.2.2 Arrays and Shapes

SOL supports two complex data structures:

Array

An array is a fixed size sequence of primitives or objects of a given *shape*. Individual elements can be accessed within array by specifying indices as integers in range `0` to `1 less than array length`.
Example:

```
func main() {
    int [3]a;       /* declare array of 3 integers */
    a = [1, 2, 3]; /* init array with 3 elements*/
    a[1] = 4;       /* update second element value to 4 */
    a[2] = a[0];    /* last element equal to first element */
}
```

Shape

A collection of one or more variables/arrays of *primitives and/or shapes* which come together to describe a *shape*. These variables are called *member variables* of a shape. A shape definition can also contain function definitions, referred as *member functions*.
Example:

```
class Line {
    string name;  /* identify a line by name */
    int [2]start; /* first end point of a line */
    int [2]end;   /* other end point of a line */

    /* compulsory constructor for a shape */
    construct(int [2]s, int[2]e) {
        /* constructor can be empty definition */
        start = s;
        end = e;
    }

    /* compulsory draw member function.*/
    draw() {
        /* accepts no arguments */
        /* can be empty definition */
    }
}

func main() {
    Line l; /* declare a variable of shape Line */
    l = shape Line([1,1], [2,2]); /* instantiate a Line object */
}
```

2.2.3 Operators

SOL supports arithmetic operations, relational comparisons and logical operations. For all binary operators described in this section, the operands are specified to the left and right, respectively, of the operator and *both operands must be expressions of same data types.*
All logical operators accept *boolean logic expressions*,represented as *integer expression* in SOL. Non-zero expressions correspond to `true` and a `zero` corresponds to `false`.

1. *Binary arithmetic operators*: `+, -, *, / %`
2. *Relational operators (binary)*: `==, |=, >, >=, <, <=`
3. *Logical operators*: `&&, ||` *(operands must be of type int)*
4. *Logical not*: `!` *(unary and right associative)*
5. *Unary negation*: `-` *(unary and right associative)*

All expressions are evaluated left to right. Please refer to section 3.5.1 for exact order of preference and associativity rules for each operator.

Example:

```
func main() {
    int x;
    int y;
    int c;
    float f;
    float g;

    x = 2;
    y = 3;
    f = 3.0;
    g = 6.0;

    y = x + y;  /*5: integer addition */
    x = y - x;  /*3: subtraction */

    g = g * f;  /* 18: floating point multiplication*/
    f = g / f;  /* 6: floating point division */
    y = y % 2;  /*1: modulo operation */

    c = g == f; /*0: EQUALITY is false */
    c = g != f; /*1: NOT EQUAL is true */
    c = g > f;  /*1: g GREATER THAN f */
    c = y >= x; /*1: y GREATER THAN OR EQUAL x is true */
    c = y < x;  /*0: y LESS THAN x is false */
    c = 5 <= 5; /*1: 5 LESS THAN OR EQUALS 5 is true */

    c = 5 && 0; /*0: LOGICAL AND of true and false */
    c = 2 || 0; /*1: LOGICAL OR of true and false */

    c = !c;   /*0: LOGICAL NOT of true(1) */
    f = -f;   /*-6: unary negation of arithmetic expression */
}
```

2.2.4 Control flow statements

SOL program statements are executed in order, with the entry point being the main function. However, sometimes developers need to execute only a branch of source code (jump through some statements) or execute a portion of code repeatedly. SOL provides two control flow statements if and while for conditional branching and looping through a portion of code, respectively.

if-statement

An *if statement block* allows to execute or skip a code branch based on a *logical expression.* Example:

```
func main() {
  if (1 > 2) {
    /* condition if false; skip this code block */
    consolePrint("NOT PRINTED");
  }
  consolePrint("Hello World");
}
```

while-statement

A while statement block allows to execute a portion of source code repeatedly until a logical

```
func main() {
    int i;
    i = 1;
    while (i <= 5) {
        consolePrint("Hello");  /* this loop prints Hello 5 times*/
        i = i + 1;     /* loop terminates when i exceeds 5 */
    }
}
```

2.2.5 Functions in SOL

In SOL functions can be defined as a way to abstract away and re-use a code lock, with a named representation. These are useful if a particular piece of code needs to be executed at multiple places in the program. Functions in SOL can be defined as stand alone functions or as *member functions* of a *shape* definition. Function definitions begin with the `func` keyword and they optionally accept arguments as input values and return a result, for which the result type needs to be indicated in the function definition. A function may return no value *(void type)* in which case no return type needs to be mentioned during function definition. Please refer section 3.6.4 for a detailed explanation of function definition syntax.

Example:

```
/* define a function that accepts two integers and returns their
   sum */
```

```
func int add(int x, int y) {
    /* sum of two integers */
    return x + y;
}

func main() {
    int sum;
    sum = add(2, 3);
    if (sum == 5) {
        consolePrint("CORRECT");
    }
}
```

Functions in SOL also support recursion. SOL also provides a number a of type conversion functions and built-in functions for displaying and animating shapes on screen and printing text on screen or console. Please refer to section 3.8 for detailed information, syntax on these functions.

2.3 Dancing Bars Tutorial (Sample SOL program)

The following program shows a simple definition of a *dancing line* as a *shape* and uses this definition to create a set of colored bars, that oscillate on one endpoint, at a given frequency.

1. As SOL treats all animation as an interaction of *shapes*, we first define a thick line *shape* in SOL, which can oscillate in length on one end.
   The constructor accepts four main arguments: `start point`, `end point`, `color` and `frequency` of oscillation
   We define its `draw` function based on these input arguments.

```
shape DanceLine {
    int [2]start;
    int [2]end;
    int [3]color;
    int freq;   /* oscillation frequency */
    int cnt;
    int d;      /* length change per iteration */

    /* constructor */
    construct(int [2]s, int [2]e, int[3]clr, int f) {
        start = s;
        end = e;
        color = clr;
        freq = f;
        cnt = 0;    /* initial counter */
        d = 2;
    }

    /* drawing specification for single frame */
```

```
    draw(){
        int i;
        int [2]s;    /* control points for drawCurve */
        int [2]m;
        int [2]e;

        /* increment count on each frame */
        cnt = cnt + 1;

        if (cnt > freq) {
            d = -d;      /* reverse direction */
            cnt = 0;     /* reset freq counter */
        }

        /* change object length on one end */
        end[1] = end[1] + d;

        s = start;  /* end points of line */
        e = end;

        /* draw 10 bezier curves for thickness */
        i = 0;
        while (i < 10) {
            s[0] = s[0] + 1;
            e[0] = e[0] + 1;

            /* bezier curve mid point */
            m[0] = (s[0] + e[0]) / 2;
            m[1] = (s[1] + e[1]) / 2;

            /* draw straight bezier curve */
            drawCurve(s, m, e, 2, color);
            i = i + 1;
        }
    }
}
```

2. The next step is to create a collection of multiple *DanceLine* instances, say *DiscoBar*. Hence, we define a new `shape`, *DiscoBars* that defines three variables of `shape` *DanceLine*. In the constructor we instantiate the *member variables* with different colored lines oscillating at different frequencies.

```
shape DiscoBars {
    DanceLine d1;    /* member variables */
    DanceLine d2;
    DanceLine d3;
```

```
    construct () {
        /* instantiate DanceLine member variables */
        d1 = shape DanceLine([100,300], [100,252],
           [30,144,255], 20);
        d2 = shape DanceLine([130,300], [130,202],
           [210,105,30], 40);
        d3 = shape DanceLine([160,300], [160,132], [50,205,50],
            80);
    }

    draw(){
        /* empty draw function
         * draw functions of member variables
         * are called implicitly
         */
    }
}
```

3. As a final step, we need to create one object of *DanceBards* shape in the *main* function, which is the entry point to a SOL program. The SOL runtime will recursively call its *draw* function and `draw` functions of its member variables at each frame, to display the osciallating lines.

```
func main() {
    DiscoBars bars;
    bars = shape DiscoBars();
}
```

Your final output on screen should look like the image shown below, with the colored bars **oscillating in length, at their upper ends**, at different frequencies.

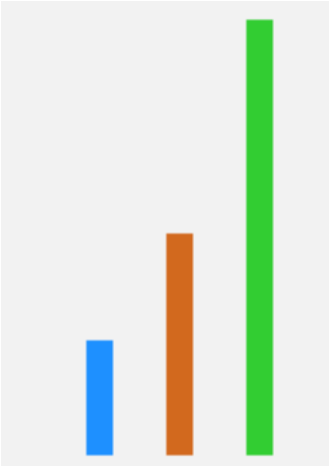


Fig 2.1 DiscoBars Tutorial

Chapter 3

Language Reference Manual

3.1 Introduction

SOL is a simple language that allows programmers to create 2D animations with ease. Programmers will have the ability to define and create objects, known as shapes, and dictate where they appear, and how they move. SOL uses *point* (visual dot) and Bézier Curves with three control points as the basic drawing controls provided to the programmer for defining shapes. As a lightweight object-oriented language, SOL allows for unlimited design opportunities and eases the burden of animation. In addition, SOLs simplicity saves programmers the trouble of learning complicated third-party animation tools, without sacrificing control over behavior of objects.

3.2 Conventions

The following conventions are followed throughout this SOL Reference Manual.

1. `literal` - Fixed space font for literals such as commands, functions, keywords, and programming language structures.
2. *variable* - The variables for SOL progamming language and words or concept being defined are denoted in italics.

The following conventions are applied while drawing and animating objects, using internal functions (see Section 3.8):

1. The origin of the drawing canvas is on the top left of the screen.
2. The positive X-axis goes from left to right.
3. The positive Y-axis goes from top to bottom.
4. Positive angles specify rotation in a clockwise direction.
5. Coordinates are specified as integer arrays of size 2, consisting of an X-coordinate followed by a Y-coordinate.
6. Colors are specified as integer arrays of size 3, consisting of Red, Green and Blue values in the range 0 - 255, where `[0, 0, 0]` is black and `[255, 255, 255]` is white.

3.3 Lexical Conventions

This section describes the complete lexical conventions followed for a syntactically correct SOL program, forming various parts of the language.

3.3.1 *Comments*

Comments in SOL start with character sequence `/*` and end at character sequence `*/`. They may extend over multiple lines and all characters following `/*` are ignored until an ending `*/` is encountered.

3.3.2 *Identifiers*

In SOL, an identifier is a sequence of characters from the set of english alphabets, arabic numerals and underscore (_). The first character of an identifier should always be a lower case english alphabet. Identifiers are case sensitive. Identifiers cannot be any of the reserved keywords mentioned in section 3.3.3.

3.3.3 *Keywords*

Keywords in SOL include data types, built-in functions, and control statements, and may not be used as identifiers as they are reserved.

int	if	main	shape
float	while	setFramerate	parent
char	func	getFramerate	extends
string	construct	print	
	return	consolePrint	
		intToString	
		floatToString	
		charToString	
		render	
		wait	
		drawPoint	
		drawCurve	
		translate	
		rotate	

3.3.4 *Integer Constants*

A sequence of one or more digits representing a number in base-10, optionally preceded by a unary negation operator (`-`), to represent negative integers.
Eg: `1234`

3.3.5 *Float Constants*

Similar to an integer, a float has an *integer*, a decimal point (`.`), and a fractional part. Both the integer and fractional part are a sequence of one or more digits. A negative float is represented

by a preceding unary negation operator (`-`).
Eg: `0.55 10.2`

3.3.6 *Character Constants*

An ASCII character within single quotation marks.
Eg: `'x' 'a'`

3.3.7 *Escape Sequences*

The following are special characters represented by escape sequences.

Name	Escape
newline	\n
tab	\t
backslash	\\
single quote	\'
double quote	\"
ASCII NUL character	\0

3.3.8 *String constants*

A SOL *string* is a sequence of zero or more *characters* within double quotation marks.
Eg: `"cat"`

3.3.9 *Operators*

SOL has mainly four categories of operators defined below:

Assignment Operator

The right associative *assignment operator* is denoted by the (`=`) symbol having a variable identifier to its left and a valid expression on its right. The *assignment operator* assigns the evaluated value of expression on the right to the variable on left.

Unary Negation Operator

The right associative unary negation operator (`-`) can be used to negate the value of an arithmetic expression.

Arithmetic Operators

The following table describes `binary arithmetic operators` supported in SOL which operate on two `arithmetic expressions` specified before and after the operator respectively. The said expressions must both be of type `int` or `float`. Please refer to section 3.5.1 for precedence and associativity rules.

Operator	Definition
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo

Comparison Operators

The comparison operators are left associative binary operators for comparing values of operands defined as expressions. Please refer to section 3.5.1 for precedence and associativity rules.

Operator	Definition
==	Equality
!=	Not Equals
<	Less than
>	Greater than
<=	Less than or equals
>=	Greater than or equals

Logical Operators

The logical operators evaluate boolean expressions and return an integer as result - with 0 as False and 1 as True. Please refer to section 3.5.1 for precedence and associativity rules.

Operator	Definition
&&	AND
||	OR
!	NOT

3.3.10 *Punctuators*

The following symbols are used for semantic organization in SOL:

Punctuator	Usage
{}	Used to denote a block of code. Must be present as a pair.
()	Specifies conditions for statements before the subsequent code, or denotes the arguments of a function. Must be present as a pair.
[]	Indicates an array. Must be present as a pair.
;	Signals the end of a line of code.
,	Used to separate arguments for a function, or elements in an array definition.

3.4 Identifier Scope

3.4.1 *Block Scope*

Identifier scope is a specific area of code wherein an identifier exists. A scope of an identifier is from its declaration until the end of the code block within which it is declared.

3.4.2 *File Scope*

Any identifier (such as a variable or a function) that is defined outside a code block has file scope i.e. it exists throughout the file.
If an identifier with file scope has the same name as an identifier with block scope, the block-scope identifier gets precedence.

3.5 Expressions and Operators

3.5.1 *Precedence and Associativity*

SOL expressions are evaluated with the following rules:

1. Expressions are evaluated from left to right, operators are left associative, unless stated otherwise.
2. Expressions within parenthesis take highest precedence and evaluated prior to substituting in outer expression.
3. The unary negation operator (`-`) and logical not operator (`!`) are placed at the second level of precedence, above the binary, comparison and logical operators. It groups right to left as described in section 3.3.9.
4. The third level of precedence is taken by multiplication (`*`), division (`/`) and modulo (`%`) operations.
5. Addition (`+`) and subtraction (`-`) operations are at the fourth level of precedence.
6. At the fifth level of precedence are the comparison operators: `<`, `>`, `<=`, `>=`.
7. At sixth level of precedence are the equality comparison operators: `==` and `!=`.
8. The logical operators, OR (`||`) and AND (`&&`) take up the next level of precedence.
9. At the final level of precedence, the right associative assignment operator (`=`) is placed, which ensures that the expression to its right is evaluated before assignment to left variable identifier.

3.5.2 *Dot Accessor*

To access members of a declared `shape` (further described in section 3.11), use the dot accessor '.'.
Eg:
```
shape_object.point1 /* This accesses the variable point1 within the object shape_object
*/
```

3.6 Declaring Identifers

Declarations determine how an identifier should be interpreted by the compiler. A declaration should include the identifier type and the given name.

3.6.1 *Type Specifiers*

SOL provides four type specifiers for data types:

- *int* - integer number
- *float* - floating point number
- *char* - a single character
- *string* - string (ordered sequence of characters)

3.6.2 *Declaring Variables*

An identifier, also referred to as a *variable* is declared by specifying the `sprimitive type` or name of `Shape`, followed by a valid identifier, as specified in section 3.3.2. Variables can be declared only at the beginning of a function or at the top of the source files, as global variables, which are accessible within all subsequent function or shape definitions.

3.6.3 *Array Declarators*

An array may be formed from any of the primitive types and `shape`s, but each array may only contain one type of primitive or `shape`. At declaration, the type specifier and the size of the array must be indicated. The array size need not be specified for strings, which are character arrays. SOL supports `fixed size arrays`, declared at compile time i.e. a program can not allocate dynamically sized arrays at runtime. Arrays are most commonly used in SOL to specify coordinates with two integers or drawing colors in RGB format with a three element array.
Eg: `int[2] coor; /* Array of two integers */`

3.6.4 *Function Declarators and Definition*

Functions are declared with the keyword: `func`. This is followed by the *return type* of the function. If no return type is specified, then the function automatically does not return any value. Functions are given a name (a valid *identifier*) followed by function formal arguments. These arguments are a comma-separated list of variable declarations within parentheses. Primitives are passed into functions by value, and objects and arrays are passed by reference. This function declaration is then followed by the function definition, within curly braces; functions must always be defined immediately after they are declared.
Functions can also be defined within *shape* definitions in which case they are referred as *member functions* of a class. (see section 3.11)

Example:

```
func example(int a, int b){
    /* a function named example that takes
       two arguments, both of type int */
}
```

3.6.5 *Constructor Declarators*

Constructors are declared with the keyword: `construct`. Constructor definitions are similar to a function definition with three additional constraints:

1. Constructors are defined inside the class definition
2. A construct is defined with `construct` keyword, followed by optional formal arguments, within parenthesis as a comma-separated list of variable declarations, similar to function definitions
3. Constructors do not have a return type specified

Example:

```
shape Point {
    int [2]coordinate;
    construct (int x, int y) {
        /* constructor definition */
        coordinate[0] = x;
        coordinate[1] = y;
    }
}
```

Please see section 3.11 for defining shapes in SOL and creating shape instances.

3.6.6 *Definitions*

A definition of a primitive type variable includes a value, assigned by the assignment operator '='. For defining arrays, `rvalue` is the sequence of array literals within square brackets. *Shapes* are objects which are initialized by calling the `construct`, with optional parameters (see section 3.11). In SOL programs, all variables *must* be *declared* before assigning values.
Example:

```
char y;         /* declarations */
float z;
int [3]w;       /* array declaration */
string s;
Triangle t;

y = 'b';        /* definitions */
z = 3.4;
w = [5, 2, 0];
s = "cats";
t = shape Triangle();    /* a triangle object */
```

3.7 Statements

A statement in SOL refers to a complete instruction for a SOL program. All statements are executed in order of sequence. The four types of statements are described in detail below:

3.7.1 *Expression Statement*

Expression statements are those statements that get evaluated and produce a result. This can be as simple as an assignment or a function call.
Eg: `x = 5; /* assign 5 to identifier x */`

3.7.2 *If Statement*

An *if* statement is a conditional statement, that is specified with the `if` keyword followed by an *expression* specified within a pair of parenthesis; further followed by a block of code within curly braces. The code specified within the `if` block executes if the expression evaluates to a non-zero *integer*.
Example:

```
int x;
x = 1;
if (x == 1) {
    /* This code gets executed */
}
```

3.7.3 *While Statement*

A *while* statement specifies the looping construct in SOL. It starts with the `while` keyword, followed by an expression specified within a pair of parenthesis; this is followed by a block of code within curly braces which is executed repeatedly as long as the condition in parentheses is valid. This condition is re-evaluated before each iteration and the code within `while` block executes if the condition evaluates to a non-zero *integer*.
Example:

```
int x;
x = 5;
while (x > 0) {
    /* This code gets executed 5 times */
    x = x - 1;
}
```

3.7.4 *Return statement*

Stops execution of a function and returns to where the function was called originally in the code. Potentially returns a value; this value must conform with the return type specified in the function declaration. If no return type was specified, a *return* statement without any value specified is syntactically valid (but not compulsory).
Example:

```
func int sum(int x, int y) {
    /* return sum of two integers */
    return x + y;
}
```

3.8 Internal Functions

SOL specifies a set of required/internal functions that must be defined for specific tasks such as drawing, rendering or as an entry point to the program, described below.

3.8.1 *main*

Every SOL program must contain a `main` function as this is the entrypoint of the program. The `main` function may, declare and define variables or shape objects or call other functions written in the program. The `main` function does not take inputs as SOL programs do not depend on user input.
Example:

```
func main() {
    /* Entry point for SOL programs */
    int x;  /* variable declaration */
    x = 1;  /* assign value */
    consolePrint("Hello World"); /* call function */
}
```

Arguments: None

3.8.2 *setFramerate*

Call `setFramerate` to specify frames per second to render on screen. The frame rate is specified as a *positive integer argument* and returns `0` for success and `-1` to indicate failure. By default, frame rate is set to 30 frames per second for a SOL program.
Arguments: `rate (int)`
Return: `0 for success, -1 for failure`

3.8.3 *getFramerate*

Call `getFramerate` to get the current number of frames rendered per second as *integer*.
Arguments: `None`
Return: `frames per second (int)`

3.8.4 *consolePrint*

Prints a string to the console. Com]monly used to print error messages.
Arguments: `text (string)`

3.8.5 Type Conversion Functions

SOL provides following type conversion functions for converting expressions of a given type to expression of another type.

intToString

Accepts an expression (`src`) of type `int` as the argument and returns the `string` representation of evaluated result.
Argument: `src (int)`
Return: value of type `string`

floatToString

Accepts an expression (`src`) of type `float` as the argument and returns the `string` representation of evaluated result.
Argument: `src (float)`
Return: value of type `string`

charToString

Accepts an expression (`src`) of type `char` as the argument and returns the `string` representation of evaluated result.
Argument: `src (char)`
Return: value of type `string`

3.9 Drawing Functions

The following set of functions are also a category of internal/required functions, which describe the drawing aspects for `shape` objects defined in a SOL program.

3.9.1 *drawPoint*

Draws a point at a specified coordinate in the specified color.
Arguments: `pt (int[2]), color (int[3])`

3.9.2 *drawCurve*

`drawCurve` is one of the basic internal functions used to draw a Bézier curve. SOL defines all possible shapes as a collection of Bézier curves. The function arguments in order are, the *three control points* for the curve, a *step size* to define smoothness of curve, and the *color* of curve in RGB format.
Arguments: `pt1 (int[2]), pt2 (int[2]), pt3 (int[2]), steps(int), color (int[3])`

3.9.3 *print*

Displays horizontal text on the render screen at the coordinates specified by the user, in specified color.
Arguments: `pt (int[2]), text (string), color (int[3])`

3.9.4 *draw*

For every `shape` definition `draw` is a required function that must be defined by the programmer. The `draw` function does not accept any input arguments and called internally to display the object on screen. The `drawCurve`, `drawPoint` and `print` functions calls can be used within `draw` definition to describe the actual drawing of an object.
At runtime `draw` functions of all objects instantiated at runtime are called, to create the final scene rendering on screen.

3.10 Animation Functions

The following functions are used to animate the objects drawn in a SOL program.

3.10.1 *translate*

Displaces a `shape` by specifying a two-element array of integers, where the first element is the number of pixels along the horizontal axis and the second element along the vertical axis, over a specified time period in seconds.
Arguments: `displace (int[2]), time (int)`

3.10.2 *rotate*

Rotate a `shape` around an axis point by a specified number of degrees over a time period in seconds.
Arguments: `axis (int[2]), angle (float), time (float)`

3.10.3 *render*

Specify the set of motions to be animated. This code-block can be defined for shapes that need to move or can be left undefined for non-moving shapes. Within this function, various `rotate` and `translate` calls can be made to move the shape. This should be specified in the `main` function.
Arguments: None

3.10.4 *wait*

Pauses animation for a specified amount of time (in seconds). To be called in the `render` function.
Arguments: `time (float)`

3.11 Classes

SOL follows an object-oriented paradigm for defining objects (drawn `shape`s) which can be further animated using the animation functions described in Section 3.10.

3.11.1 *shape*

Similar to a class in C++; a *shape* defines a particular 2-D shape as part of the drawing on screen. The name of a *shape* must always start with an uppercase english alphabet.

Shape definition

A *shape* definition starts with the `shape` keyword, followed by the *shape name*,(eg: `Triangle`) and the definition within curly braces ({}) code block. Shape definitions may optionally contain *member variables*.

Every *shape* must define a *constructor* using `construct` keyword and `draw` function. The `construct` definition can optionally have formal arguments as input parameters. The `draw` function does not accept any arguments and its definition can have multiple `drawPoint`, `drawCurve` and `print` function calls to describe on screen display of the object.

It is possible to define *functions* in a shape definition. The member functions are defined with the same rules as specified in section 3.6.4.
When *member variables* are accessed within a member function, it is implied that the member variables belong to the current object that calls the function. If a *member variable* or *global variable* name is same as that of a local variable or *formal argument* in function definition, then the *local variable* or *formal argument* overshadows the other conflicting variable.

Example:

```
shape Triangle {
    int[2] a; /* Corners of a triangle */
    int[2] b;
    int[2] c;

    int[2] p;  /* mid points of lines*/
    int[2] q;
    int[2] r;

    construct (int [2]x, int [2]y, int [2]z) {
        a = x;
        b = y;
        c = z;

        findCenter(p, a, b);
        findCenter(q, b, c);
        findCenter(r, a, c)
    }

    /* internal draw function definition */
    draw() {
        /* Draw triangle lines with bezier curves */
        drawcurve(a, p, b, 100, [255,0,0]); /*red*/
        drawcurve(b, q, c, 100, [0,255,0]); /*green*/
        drawcurve(c, r, a, 100, [0,0,255]); /*blue*/
    }

    /* write result in pre-allocated array res */
```

```
    func findCenter(int[2]m, int[2]x, int[2]y){
        m[0] = (x[0] + y[0]) / 2;
        m[1] = (x[1] + y[1]) / 2;
    }
}
```

Creating Shape Instances

Actual instances for a shape definition can be created, which represent the actual shapes rendered on the screen.
To instantiate an object for a shape, we first declare a variable of defined *shape* (say `Triangle`) and then instantiate it by calling the constructor.
Example:

```
func main() {
    /* declare variable of shape Triangle */
    Triangle t;

    /* instantiate a triangle */
    t = shape Triangle([100,100], [200,100], [150,200]);
}
```

3.11.2 *Inheritance*

SOL allows single class inheritance for shapes i.e given a shape, such as `Line`, one may create a sub-shape of `Line`, called `LineBottom`, and inherit all of its fields from the parent `shape`, `Line`, using the keyword `extends`.
Example:

```
shape Line {
    int[2] a;
    int[2] b;

    construct (int[2] a_init, int[2] b_init) {
        int i;
        i = 0;
        /* copy values */
        while (i < 2) {
            a[i] = a_init[i];
            b[i] = b_init[i];
            i = i + 1;
        }
    }

    func findCentre(int[2] res, 2int[2] x, int[2] y) {
        /* write result to res */
        int i;
```

```
        i = 0;
        while(i < 2) {
            res[i] = (a[i] + b[i]) / 2;
            i = i + 1;
        }
    }

    func draw() {
        drawcurve(a, findCentre(a, b), b, [0, 0, 0]);
    }
}

/* Subclass of Line */
shape LineBottom extends Line {
    int[2] c;
    int[2] d;

    construct (int[2] a_init, int[2] b_init, int[2] c_init) {
        parent(a_init, b_init);
        c = c_init;
        d = b;
    }

    func draw() {
        parent();
        drawcurve(c, findCentre(c, d), d, [0, 0, 0]);
    }
}
```

parent (keyword)

The parent `shape`'s functions can be accessed by the function call `parent()`. This invokes the implementation of the current member function defined in the parent `shape`. In constructors, the `parent()` calls the constructor for the parent `shape`.

Chapter 4

Project Plan

This chapter details the project development, testing process, timelines, milestones and roles and responsibilities of each team member.

4.1 Project Development Process

Throughout the SOL compiler development process we tried to stick as closely as possible to in-lecture instructions and suggestions given by instructor, Prof. Stephen A. Edwards, and our project mentor Connor Abbott. The four major phases of project development are described in the following sections.

4.1.1 Planning

Once the team decided on addressing easy animations through an object-oriented programming language, we quickly got down to listing as many features as we could think of with respect to a graphics animation. As Aditya Narayanamoorthy *(Language Guru)* had taken a Computer Graphics course previously, he helped us pick out the most basic and necessary features for the language, such as using *Bézier curves* to describe shapes in our language. We tried to narrow down the minimal number of language features that would allow a SOL programmer to describe, display and animate a shape on screen using SOL. In case of a disagreement on the features to be included, the final call was taken by the *Language Guru*. All the decided features are detaile in the SOL Language Reference Manual (chapter 3), which every team member used as a reference point for the rest of the project development and testing phases.

In our planning phase we also decided on the software development process, including division of work based on each member's expertise, discussed time availabilities, tools and development environments to use, summarized below:

1. We scheduled two days per week for 3 hour long meetings for the team to work together on the project.
2. Chose Github as the version control system for our project, in conjunction with Slack for team communication.

4.1.2 Specfication

For specifying the exact feature details, we required a lightweight graphics library, that would interface with the graphics backend and link well against the LLVM compiler output and develop a good understanding of its working. Erik, the systems architect, tried out a few graphics library and suggested using SDL2, primarily because of its ease of use for rendering simple graphics on screen and it's bindings of multiple languages including C, C++ and OCaml.

The next major specification step for SOL programming language, was to decide on how to represent and render shapes. Aditya, our Language Guru, suggested that all shapes can be expressed as a combination of Bézier curves and hence the basic units of drawing in SOL, available to a programmer should be the `drawPoint` and `drawCurve` functions to draw shapes, along with `translate` and `rotate` functions to animate the drawings in a 2D plane. To accomplish drawing Bézier curves, on top of SDL2, we chose the SDL2_gfx library which was compiled and integrated into the development environment by Kunal and Erik.

Throughout the compiler development, we asked our mentor Connor Abbott for advice on how to prioritize and implement features crucial for SOL, like fixed size arrays v/s dynamic arrays or advantages of using an SAST over a simple AST. During the development phase we also realized that some of the features that we had originally planned for SOL were not quite straightforward to implement, or differed from our initial understanding. This made us modify some specific components of the language, which we updated in the Language Reference Manual and testing suite, after consulting with our mentor.

4.1.3 Development and Testing

We followed a *test driven development* approach for SOL language as far as possible. Initially test cases were written based on the features mentioned in our programming language reference manual. Test cases were divided into two main categories, features that are allowed in SOL, such as integer addition, defining shapes etc and failure cases which check whether the compiler properly rejects syntactically or semantically incorrect programs. Each test case comprised of a basic SOL program source code that would test one particular feature or component of the language and compare the actual output against an standardized file containing expected output messages.

The compiler was built by integrating one feature at a time, end to end, starting at the scanner, all the way up to codegen phase and which would allow the compiler to support that feature alongside previously implemented features. Each feature implementation was tested against the test suite to ensure correct behaviour and also ensure that it does not breaks any of the previously implemented features, using the testing script and Travis CI builds that were triggered on each code push to our github repository.
Aditya and Gergana worked heavily on the compiler and met frequently outside of scheduled team meetings to pair-program and ensure on-time delivery of components.

Towards the end of project development timeline, we also incorporated a manual testing component, as there is no way to ensure correct display of graphics within Travis CI environment. These tests can be run using the same test suite script, with manual inspection of displayed graphics to ensure correctness.

4.2 Programming Style Guide

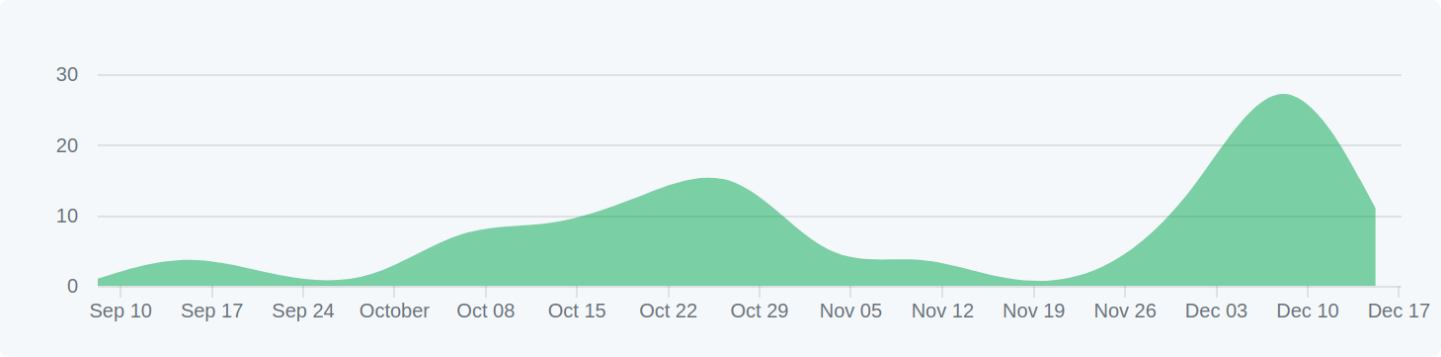
- Most of our compiler source code is written in OCaml, we simply used the standard Ocaml programming guidelines as mentioned on the official tutorials page
  https://ocaml.org/learn/tutorials/guidelines.html
- To enforce OCaml style guides, Kunal set up the open sourced merlin linter and analyzer in sublime text editor
  https://github.com/let-def/sublime-text-merlin
- A significant component of SOL compiler is the *predefined.o* static library written in C (C99 standard) for which the C++ linter was used within sublime text.
  https://github.com/SublimeLinter/SublimeLinter-cppcheck
- As a general rule of thumb, we tried to use descriptive variable names throughout our code, which helps the reader to easily understand the code.

4.3 Project Timeline

4.3.1 Milestones

Milestone	**Date**
Initial research & brainstorming	Sept. 11
Decide on language idea	Sept. 19
Finish Proposal	Sept. 26
Finish LRM	Oct. 16
Add regression test suite/Travis CI	Oct. 29
Implement primitives & other basics	Nov. 1
Hello World Demo	Nov. 8
Implement arrays, internal functions	Dec. 2
Implement shapes	Dec. 12
Implement drawing (SDL)	Dec. 16-17
Update and Finish Testing (manual)	Dec. 17-19
Final Report Preparation	Dec. 18-19
Demo Day	Dec. 20

4.3.2 Github timeline



4.4 Team Roles and Responsibilities

The following table summarizes the major responsibilities taken up by each team member, with a number of components overlapping among team members.

Team Member	**Responsibilities**
Aditya Narayanamoorthy (Language Guru)	Scanner, Parser, Ast, Semant, Sast, Codegen
Gergana Alteva (Project Manager)	Scanner, Parser, Ast, Semant, Sast, Codegen
Erik Dyer (System Architect)	VM, Docker, Graphics Library researching (SDL2), testing
Kunal Baweja (Tester)	predefined library, SDL2, SDL2_gfx, Automated and Manual Test suite, Travis CI

4.5 Software Development Environment

The following software development tools, libraries and dependencies were used for developing SOL compiler by the team.

1. Ubuntu `15.04` VM setup with ocaml compiler, opam, llvm toolchain, and ocaml Llvm module for llvm bindings.
2. Ocaml version: `4.02.3`
3. LLVM dev library: `3.8`
4. opam Llvm package (Ocaml Llvm bindings)
5. Sublime text with `merlin(ocaml)` and `cppcheck(C99)` linters
6. Version Control: `Github`
7. Travis CI: Automated testing on `Ubuntu 16.04` environment (same dependency configurations)

4.6 Project Logs

Please see appendix E for project logs in reverse chronological order of timeline.

Chapter 5

Architectural Design

We describe the compilation structure for a SOL program through SOL compiler in this chapter. This can be broken down into 3 phases:

1. Compile SOL program to LLVM IR bytecode representation
2. Compile generated IR to architecture specific assembly code using LLVM compiler
3. Compile and link against predefined static library to produce executable

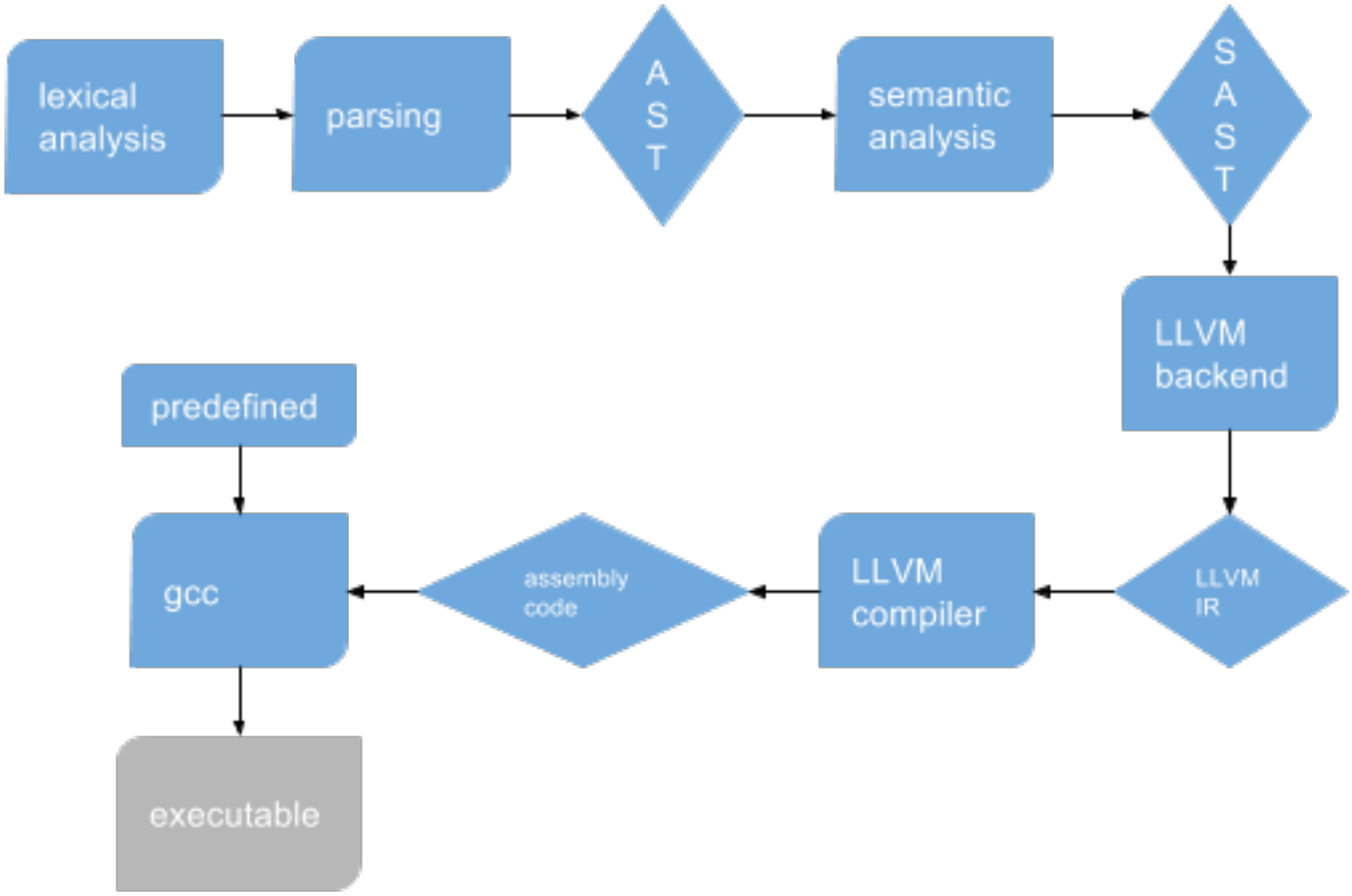


Figure 5.1: SOL Compiler Block Diagram

5.1 Interfaces between SOL Compiler components

5.1.1 Scanner

The scanner takes as input a `SOL(.sol) source program` and generates tokens for *identifiers, keywords, operators, values, functions and shapes*, as specified by the lexical conventions for SOL in Language Reference Manual (see chapter 3). It rejects a source program if it observes an invalid token, against the rules specified in `scanner.mll`

5.1.2 Parser

The parser accepts the tokenized output from scanner and generates an abstract syntax tree (AST) based on SOL syntax. It rejects the source program if a syntactic error is seen due to which a valid AST can not be generated.

5.1.3 Semantic Checker

The AST output from parser is passed to semantic checker to ensure semantic correctness of each program statement, including static type checking. Eg: An `int` variable can not be assigned a floating point value, per SOL specification. Such an assignment in SOL program will be semantically incorrect and rejected at the semantic checking phase.
The semantic checking phase associates semantic information to the AST nodes and produces a SAST, if the program passes the semantic check.

5.1.4 Code Generation

The code generator accepts the SAST generated after semantic checking and generates the LLVM IR code using the OCaml Llvm bindings. The SOL *primitives, arrays, shapes, functions, member functions* are all mapped to corresponding LLVM primitives, arrays, structs and function definitions representations.
Once the LLVM IR code is generated successfully, the SOL compiler task is over. After this we use the LLVM compiler (`llc`) to convert the IR code to architecture specific assembly code. This is further compiled into an executable and statically linked against a predefined library, that interfaces with `SDL2` and `SDL2_gfx`.

The compiler components were implemented by team members as listed below:

Component	**Source file**	**Team members**
Lexical Analysis	scanner.mll	Aditya, Gergana
Parsing	parser.mly	Aditya, Gergana
Abstract Syntax Tree	ast.ml	Aditya, Gergana
Semantic Analysis	semant.ml	Aditya, Gergana
SAST	sast.ml	Aditya, Gergana
LLVM backend	codegen.ml	Aditya, Gergana
Predefined graphic library	predefined.c	Kunal

Chapter 6

Test Plan

The SOL compiler was developed through a test driven development approach.

6.1 Sample Programs

This section shows 3 sample programs written in SOL and corresponding LLVM IR code represtation generated for the source programs.

6.1.1 Recursive function

The following program shows a SOL program to print sum of natural numbers from 1 to `n`, where `n` is input to the program.

series.sol

```
/* @author: Kunal Baweja */

/* recursive series sum of 1 to n */
func int series(int n) {
        int x;
        if (n < 0) {
                consolePrint(intToString(0));
                return 0;
        }
        if (n < 2) {
                consolePrint(intToString(n));
                return n;
        }
        x = n + series(n-1);
        consolePrint(intToString(x));
        return x;
}

func main() {
        series(5); /*1,3,6,10,15*/
```

```
}
```

series.ll

```
; ModuleID = 'SOL'

@_Running = global i1 false
@fmt = global [4 x i8] c"%s\0A\00"
@int_fmt = global [3 x i8] c"%d\00"
@flt_fmt = global [3 x i8] c"%f\00"
@char_fmt = global [3 x i8] c"%c\00"

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

declare i32 @startSDL(...)

declare void @onRenderStartSDL(...)

declare void @onRenderFinishSDL(...)

declare i32 @stopSDL(...)

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

declare i32 @drawCurve([2 x i32]*, [2 x i32]*, [2 x i32]*, i32, [3
   x i32]*)

declare i32 @drawPoint([2 x i32]*, [3 x i32]*)

declare i32 @print([2 x i32]*, i8*, [3 x i32]*)

declare void @setFramerate(i32)

declare i32 @getFramerate()

define i32 @main() {
entry:
  %startSDL = call i32 (...) @startSDL()
  %series_result = call i32 @series(i32 5)
  br label %while

while:                                            ; preds = %
   while_body, %entry
  %_Running_val = load i1, i1* @_Running
  br i1 %_Running_val, label %while_body, label %merge
```

```
while_body:                                       ; preds = %while
  call void (...) @onRenderStartSDL()
  call void (...) @onRenderFinishSDL()
  br label %while

merge:                                            ; preds = %while
  %stopSDL_ret = alloca i32
  %stopSDL_ret1 = call i32 (...) @stopSDL()
  store i32 %stopSDL_ret1, i32* %stopSDL_ret
  %stopSDL_ret2 = load i32, i32* %stopSDL_ret
  ret i32 %stopSDL_ret2
}

define i32 @series(i32 %n) {
entry:
  %n1 = alloca i32
  store i32 %n, i32* %n1
  %x = alloca i32
  %tmp = load i32, i32* %n1
  %tmp2 = icmp slt i32 %tmp, 0
  br i1 %tmp2, label %then, label %merge

merge:                                            ; preds = %entry
  %tmp3 = load i32, i32* %n1
  %tmp4 = icmp slt i32 %tmp3, 2
  br i1 %tmp4, label %then6, label %merge5

then:                                             ; preds = %entry
  %intToString = alloca i8, i32 12
  %intToStringResult = call i32 (i8*, i8*, ...) @sprintf(i8* %
     intToString, i8* getelementptr inbounds ([3 x i8], [3 x i8]*
     @int_fmt, i32 0, i32 0), i32 0)
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
     ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i8* %intToString)
  ret i32 0

merge5:                                           ; preds = %merge
  %tmp12 = load i32, i32* %n1
  %tmp13 = load i32, i32* %n1
  %tmp14 = sub i32 %tmp13, 1
  %series_result = call i32 @series(i32 %tmp14)
  %tmp15 = add i32 %tmp12, %series_result
  store i32 %tmp15, i32* %x
  %tmp16 = load i32, i32* %x
  %intToString17 = alloca i8, i32 12
  %intToStringResult18 = call i32 (i8*, i8*, ...) @sprintf(i8* %
     intToString17, i8* getelementptr inbounds ([3 x i8], [3 x i8]*
```

```
      @int_fmt, i32 0, i32 0), i32 %tmp16)
  %printf19 = call i32 (i8*, ...) @printf(i8* getelementptr
     inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i8* %
     intToString17)
  %tmp20 = load i32, i32* %x
  ret i32 %tmp20

then6:                                            ; preds = %merge
  %tmp7 = load i32, i32* %n1
  %intToString8 = alloca i8, i32 12
  %intToStringResult9 = call i32 (i8*, i8*, ...) @sprintf(i8* %
     intToString8, i8* getelementptr inbounds ([3 x i8], [3 x i8]*
     @int_fmt, i32 0, i32 0), i32 %tmp7)
  %printf10 = call i32 (i8*, ...) @printf(i8* getelementptr
     inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i8* %
     intToString8)
  %tmp11 = load i32, i32* %n1
  ret i32 %tmp11
}
```

6.1.2 Composite Shape Drawing

The following program demonstrates construction of complex shapes in SOL. It defines two shapes, `Line` and `Rectangle`. The `Rectangle` shape definition uses 4 `Line` objects to demonstrate drawing of complex shapes by aggregating simpler shapes.

composite-square.sol

```
/* @author: Kunal Baweja */

/* Test member shapes */

/* Define a line */
shape Line {
    int [2]start;
    int [2]mid;
    int [2]end;
    int [3]color;

    construct(int [2]first, int [2]second, int [3]clr) {
        start = first;
        end = second;
        color = clr;

        mid[0] = (start[0] + end[0]) / 2;
        mid[1] = (start[1] + end[1]) / 2;
```

```
    }

    draw(){
        drawCurve(start, mid, end, 2, color);
    }

}

/* Define rectange as a collection of lines */
shape Rectangle {
    Line top;
    Line right;
    Line bottom;
    Line left;

    construct(int [2]a, int [2]b, int [2]c, int [2]d) {
        top = shape Line(a, b, [150, 0, 0]);
        right = shape Line(b, c, [0, 150, 0]);
        bottom = shape Line(c, d, [0, 0, 150]);
        left = shape Line(d, a, [150, 150, 150]);
    }

    draw(){}

}

func main() {
    Rectangle sq;

    /* define lines */

    /* initialize square */
    sq = shape Rectangle([10,10], [300,10], [300, 300], [10, 300]);
}
```

composite-square.ll

```
; ModuleID = 'SOL'

%Rectangle = type { %Line, %Line, %Line, %Line }
%Line = type { [2 x i32], [2 x i32], [2 x i32], [3 x i32] }

@_Running = global i1 false
@fmt = global [4 x i8] c"%s\0A\00"
@int_fmt = global [3 x i8] c"%d\00"
@flt_fmt = global [3 x i8] c"%f\00"
```

```
@char_fmt = global [3 x i8] c"%c\00"

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

declare i32 @startSDL(...)

declare void @onRenderStartSDL(...)

declare void @onRenderFinishSDL(...)

declare i32 @stopSDL(...)

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

declare i32 @drawCurve([2 x i32]*, [2 x i32]*, [2 x i32]*, i32, [3
   x i32]*)

declare i32 @drawPoint([2 x i32]*, [3 x i32]*)

declare i32 @print([2 x i32]*, i8*, [3 x i32]*)

declare void @setFramerate(i32)

declare i32 @getFramerate()

define i32 @main() {
entry:
  %startSDL = call i32 (...) @startSDL()
  %sq = alloca %Rectangle
  %arr_copy = alloca [2 x i32]
  store [2 x i32] [i32 10, i32 300], [2 x i32]* %arr_copy
  %arr_copy1 = alloca [2 x i32]
  store [2 x i32] [i32 300, i32 300], [2 x i32]* %arr_copy1
  %arr_copy2 = alloca [2 x i32]
  store [2 x i32] [i32 300, i32 10], [2 x i32]* %arr_copy2
  %arr_copy3 = alloca [2 x i32]
  store [2 x i32] [i32 10, i32 10], [2 x i32]* %arr_copy3
  %Rectangle_inst_ptr = call %Rectangle* @Rectangle__construct([2 x
      i32]* %arr_copy3, [2 x i32]* %arr_copy2, [2 x i32]* %
     arr_copy1, [2 x i32]* %arr_copy)
  %Rectangle_inst = load %Rectangle, %Rectangle* %
     Rectangle_inst_ptr
  store %Rectangle %Rectangle_inst, %Rectangle* %sq
  %tmp = getelementptr inbounds %Rectangle, %Rectangle* %sq, i32 0,
      i32 0
  %tmp4 = getelementptr inbounds %Rectangle, %Rectangle* %sq, i32
     0, i32 1
```

```
  %tmp5 = getelementptr inbounds %Rectangle, %Rectangle* %sq, i32
     0, i32 2
  %tmp6 = getelementptr inbounds %Rectangle, %Rectangle* %sq, i32
     0, i32 3
  br label %while

while:                                            ; preds = %
   while_body, %entry
  %_Running_val = load i1, i1* @_Running
  br i1 %_Running_val, label %while_body, label %merge

while_body:                                       ; preds = %while
  call void (...) @onRenderStartSDL()
  call void @Rectangle__draw(%Rectangle* %sq)
  call void @Line__draw(%Line* %tmp)
  call void @Line__draw(%Line* %tmp)
  call void @Line__draw(%Line* %tmp4)
  call void @Line__draw(%Line* %tmp4)
  call void @Line__draw(%Line* %tmp5)
  call void @Line__draw(%Line* %tmp5)
  call void @Line__draw(%Line* %tmp6)
  call void @Line__draw(%Line* %tmp6)
  call void (...) @onRenderFinishSDL()
  br label %while

merge:                                            ; preds = %while
  %stopSDL_ret = alloca i32
  %stopSDL_ret7 = call i32 (...) @stopSDL()
  store i32 %stopSDL_ret7, i32* %stopSDL_ret
  %stopSDL_ret8 = load i32, i32* %stopSDL_ret
  ret i32 %stopSDL_ret8
}

define %Rectangle* @Rectangle__construct([2 x i32]* %a, [2 x i32]*
   %b, [2 x i32]* %c, [2 x i32]* %d) {
entry:
  %__Rectangle_inst = alloca %Rectangle
  %top = getelementptr inbounds %Rectangle, %Rectangle* %
     __Rectangle_inst, i32 0, i32 0
  %right = getelementptr inbounds %Rectangle, %Rectangle* %
     __Rectangle_inst, i32 0, i32 1
  %bottom = getelementptr inbounds %Rectangle, %Rectangle* %
     __Rectangle_inst, i32 0, i32 2
  %left = getelementptr inbounds %Rectangle, %Rectangle* %
     __Rectangle_inst, i32 0, i32 3
  %arr_copy = alloca [3 x i32]
  store [3 x i32] [i32 150, i32 0, i32 0], [3 x i32]* %arr_copy
```

```
%tmp = load [2 x i32], [2 x i32]* %b
%arr_copy1 = alloca [2 x i32]
store [2 x i32] %tmp, [2 x i32]* %arr_copy1
%tmp2 = load [2 x i32], [2 x i32]* %a
%arr_copy3 = alloca [2 x i32]
store [2 x i32] %tmp2, [2 x i32]* %arr_copy3
%Line_inst_ptr = call %Line* @Line__construct([2 x i32]* %
   arr_copy3, [2 x i32]* %arr_copy1, [3 x i32]* %arr_copy)
%Line_inst = load %Line, %Line* %Line_inst_ptr
store %Line %Line_inst, %Line* %top
%arr_copy4 = alloca [3 x i32]
store [3 x i32] [i32 0, i32 150, i32 0], [3 x i32]* %arr_copy4
%tmp5 = load [2 x i32], [2 x i32]* %c
%arr_copy6 = alloca [2 x i32]
store [2 x i32] %tmp5, [2 x i32]* %arr_copy6
%tmp7 = load [2 x i32], [2 x i32]* %b
%arr_copy8 = alloca [2 x i32]
store [2 x i32] %tmp7, [2 x i32]* %arr_copy8
%Line_inst_ptr9 = call %Line* @Line__construct([2 x i32]* %
   arr_copy8, [2 x i32]* %arr_copy6, [3 x i32]* %arr_copy4)
%Line_inst10 = load %Line, %Line* %Line_inst_ptr9
store %Line %Line_inst10, %Line* %right
%arr_copy11 = alloca [3 x i32]
store [3 x i32] [i32 0, i32 0, i32 150], [3 x i32]* %arr_copy11
%tmp12 = load [2 x i32], [2 x i32]* %d
%arr_copy13 = alloca [2 x i32]
store [2 x i32] %tmp12, [2 x i32]* %arr_copy13
%tmp14 = load [2 x i32], [2 x i32]* %c
%arr_copy15 = alloca [2 x i32]
store [2 x i32] %tmp14, [2 x i32]* %arr_copy15
%Line_inst_ptr16 = call %Line* @Line__construct([2 x i32]* %
   arr_copy15, [2 x i32]* %arr_copy13, [3 x i32]* %arr_copy11)
%Line_inst17 = load %Line, %Line* %Line_inst_ptr16
store %Line %Line_inst17, %Line* %bottom
%arr_copy18 = alloca [3 x i32]
store [3 x i32] [i32 150, i32 150, i32 150], [3 x i32]* %
   arr_copy18
%tmp19 = load [2 x i32], [2 x i32]* %a
%arr_copy20 = alloca [2 x i32]
store [2 x i32] %tmp19, [2 x i32]* %arr_copy20
%tmp21 = load [2 x i32], [2 x i32]* %d
%arr_copy22 = alloca [2 x i32]
store [2 x i32] %tmp21, [2 x i32]* %arr_copy22
%Line_inst_ptr23 = call %Line* @Line__construct([2 x i32]* %
   arr_copy22, [2 x i32]* %arr_copy20, [3 x i32]* %arr_copy18)
%Line_inst24 = load %Line, %Line* %Line_inst_ptr23
store %Line %Line_inst24, %Line* %left
```

```
  ret %Rectangle* %__Rectangle_inst
}

define void @Rectangle__draw(%Rectangle* %__Rectangle_inst) {
entry:
  %top = getelementptr inbounds %Rectangle, %Rectangle* %
     __Rectangle_inst, i32 0, i32 0
  %right = getelementptr inbounds %Rectangle, %Rectangle* %
     __Rectangle_inst, i32 0, i32 1
  %bottom = getelementptr inbounds %Rectangle, %Rectangle* %
     __Rectangle_inst, i32 0, i32 2
  %left = getelementptr inbounds %Rectangle, %Rectangle* %
     __Rectangle_inst, i32 0, i32 3
  ret void
}

define %Line* @Line__construct([2 x i32]* %first, [2 x i32]* %
   second, [3 x i32]* %clr) {
entry:
  %__Line_inst = alloca %Line
  %start = getelementptr inbounds %Line, %Line* %__Line_inst, i32
     0, i32 0
  %mid = getelementptr inbounds %Line, %Line* %__Line_inst, i32 0,
     i32 1
  %end = getelementptr inbounds %Line, %Line* %__Line_inst, i32 0,
     i32 2
  %color = getelementptr inbounds %Line, %Line* %__Line_inst, i32
     0, i32 3
  %tmp = load [2 x i32], [2 x i32]* %first
  store [2 x i32] %tmp, [2 x i32]* %start
  %tmp1 = load [2 x i32], [2 x i32]* %second
  store [2 x i32] %tmp1, [2 x i32]* %end
  %tmp2 = load [3 x i32], [3 x i32]* %clr
  store [3 x i32] %tmp2, [3 x i32]* %color
  %tmp3 = getelementptr [2 x i32], [2 x i32]* %start, i32 0, i32 0
  %tmp4 = load i32, i32* %tmp3
  %tmp5 = getelementptr [2 x i32], [2 x i32]* %end, i32 0, i32 0
  %tmp6 = load i32, i32* %tmp5
  %tmp7 = add i32 %tmp4, %tmp6
  %tmp8 = sdiv i32 %tmp7, 2
  %tmp9 = getelementptr [2 x i32], [2 x i32]* %mid, i32 0, i32 0
  store i32 %tmp8, i32* %tmp9
  %tmp10 = getelementptr [2 x i32], [2 x i32]* %start, i32 0, i32 1
  %tmp11 = load i32, i32* %tmp10
  %tmp12 = getelementptr [2 x i32], [2 x i32]* %end, i32 0, i32 1
  %tmp13 = load i32, i32* %tmp12
  %tmp14 = add i32 %tmp11, %tmp13
```

```
  %tmp15 = sdiv i32 %tmp14, 2
  %tmp16 = getelementptr [2 x i32], [2 x i32]* %mid, i32 0, i32 1
  store i32 %tmp15, i32* %tmp16
  ret %Line* %__Line_inst
}

define void @Line__draw(%Line* %__Line_inst) {
entry:
  %start = getelementptr inbounds %Line, %Line* %__Line_inst, i32
     0, i32 0
  %mid = getelementptr inbounds %Line, %Line* %__Line_inst, i32 0,
     i32 1
  %end = getelementptr inbounds %Line, %Line* %__Line_inst, i32 0,
     i32 2
  %color = getelementptr inbounds %Line, %Line* %__Line_inst, i32
     0, i32 3
  %drawCurve_result = call i32 @drawCurve([2 x i32]* %start, [2 x
     i32]* %mid, [2 x i32]* %end, i32 2, [3 x i32]* %color)
  ret void
}
```

6.2 Test Suite

The test suite is divided into two major categories:

- Automated Tests - run on Travis CI
- Manual Tests - visually inspected for display correctness

NOTE: Please refer to Appendix C for complete code listing of test cases.

Test cases were written based on the syntactic and semantic specifications of language mentioned in SOL Language Reference Manual (chapter 3). This acted as a reference point for the team and allowed us to stay as close to the LRM specifications as practically possible throughout the compiler development process.

The initial tests for each feature were written as *functional tests* which would ensure the correct working of the feature once it was implemented and integrated into the compiler. Once the feature was integrated into compiler, it was put through the complete list of tests, automated, to ensure that the new changes did not adversely affect any of the previously implemented features and to verify that as part of the language, each individual feature works in a certain expected manner without any conflicts.

We tested the following major components of our language, crucial for SOL:

- *primitives* - The tests cover declaration and assignments of primitive variables extensively, through specific test cases for assignment, operations, declarations, scoping rules and static type checking.

- *Arrays* - Fixed size arrays of primitives are supported by SOL and hence we extensively tested for array declaration, assigning and accessing individual elements. SOL does not provide bounds checking, much like C and hence it is programmer's responsibility to ensure that.
- *Shapes* - Defining shapes for drawing and animation on screen is the basic feature of object-oriented paradigm supported by SOL. We tested defining, instantiating and animating shapes through multiple automated and manual test cases, for visual correctness of rendered drawings. This includes defining and creating complex shapes using simpler shape definitions.
- *Operations* - SOL allows basic arithmetic and logical operations on integers and floating point numbers and these have been tested with respect to order of precedence and associativity rules in an expression, alongside correctness of result output by each individual operation.
- *Functions* - SOL supports declaring functions, both standalone as well as member functions of a shape. These have been tested by checking for return types, values, recursion, expected number of arguments and argument types, syntactic correctness and expected failures in case of incorrect source programs.
- *Type Conversion* - SOL allows converting, `int`, `float` and `char` values to `string` through appropriate functions mentioned in the Language Reference Manual and tested via `consolePrint` and `print` function calls.
- *Drawing Function* - Three basic drawing functions are provided as built-ins by SOL for rendering text and shapes on screen. These have been tested through manual testcases, wherein a visual inspection of displayed content is required to ensure correctness.
- *Output Function* - Apart from the drawing functions mentioned above, the only output function provided is `consolePrint` which outputs the `string` argument to standard output and tested by comparing the outputs against expected outputs of corresponding test programs.
- *Animation Function* - The *translate* and *rotate* functions are the basic components that allow a SOL programmer to animate objects and these have been tested for intended behaviour rigorously to account for intended and possible unintended side effects of the program written.

6.3 Test Automation

We used *Travis CI* in conjunction with the `testall.sh` script (see C) for automatically running all test cases and checking outputs against the expected output from the compiled program or failure during compilation or runtime of program. Setting up Travis CI allowed us to run the complete regression test suite in a standardized development environment, for every code push to the github repository, without manual intervention and be notified in case of test failures.

6.4 Manual Testing

Testing shape display and animation correctness was not possible within Travis CI environment as it does not support a display console for testing. These test cases were written separately, in test files following tnaming pattern `mnl-*`, compiled and run using the same test script, although these required manual intervention for correctness of displayed graphics. Please see Appendix D for manual test cases.

6.5 Testing Responsibilities

Team member	**Responsibilties**
Kunal Baweja (Tester)	Test Plan, Travis CI/Automation, Test environment setup, Writing regression & unit tests
Erik Dyer (System Architect)	Writing regression & unit tests

Chapter 7

Lessons Learned

7.1 Aditya Narayanmoorthy

7.2 Erik Dyer

This semester I learned how important organization and communication is. I think one thing that was really good about our team is that we were very good about sticking to meeting frequently from the beginning. Having this organization from the beginning allowed us to avoid the ẅhoś free to meet when?d̈ance and therefore build up an early momentum that was sustained throughout the semester.

I think that one piece of advice I'd give to future groups is to think about what kind of language you could build from before you take the class. Our group took several meetings before we decided what kind of language we wanted to have, and thatś before we even got to the language details. Coming in with a language idea will definitely be helpful and save time.

7.3 Gegana Alteva

Building a compiler in a team has provided me with many lessons throughout the semester. My development experience is limited and while I had an understanding of what needed to be done, I was not sure how it should be done. I learned the importance of implementing our language vertically and how to prioritize components. My advice for future teams would be to prioritize what needs to be implemented by understanding what is absolutely needed for your language to work. There are many language design aspects that seem necessary, but in reality are just ïice-to-haveś. Had I done a better job of prioritizing and classifying what is and is not a ïice-to-have;, we would have perhaps spent more time coding what was actually needed.

7.4 Kunal Baweja

Appendices

Appendix A

SOL Compiler

Code listing for compiler code. Author names are mentioned as first comment line of each code listing.

A.1 scanner.mll

```
(* Ocamllex scanner for SOL *)

{ open Parser }

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"      { comment lexbuf }           (* Comments *)
| '('       { LPAREN }
| ')'       { RPAREN }
| '{'       { LBRACE }
| '}'       { RBRACE }
| '['       { LSQUARE }
| ']'       { RSQUARE }
| ';'       { SEMI }
| ','       { COMMA }
| '+'       { PLUS }
| '-'       { MINUS }
| '*'       { TIMES }
| '/'       { DIVIDE }
| '%'       { MODULO }
| '='       { ASSIGN }
| "=="      { EQ }
| "!="      { NEQ }
| '<'       { LT }
| "<="      { LEQ }
| ">"       { GT }
| ">="      { GEQ }
| "&&"      { AND }
| "||"      { OR }
```

```
| "!"       { NOT }
| "if"      { IF }
| "while"   { WHILE }
| "return"  { RETURN }
| "int"     { INT }
| "float"   { FLOAT }
| "char"    { CHAR }
| "string"   { STRING }
| "func"    { FUNC }
| "shape"   { SHAPE }
| "construct"   { CONSTRUCT }
| "draw"    { DRAW }
| '.'       { DOT }
| "render"   { RENDER }
(*| "parent"   { PARENT }
| "extends"   { EXTENDS }
| "main"    { MAIN }   (* Consider moving out when main needs to be a
    reserved keyword *)
| "consolePrint"   { CONSOLEPRINT }
| "length"   { LENGTH }
| "setFramerate"   { SETFRAMERATE }
| "translate"   { TRANSLATE }
| "rotate"   { ROTATE }
| "wait"   { WAIT }*)
| ['0'-'9']+'.'['0'-'9']+ as lxm { FLOAT_LITERAL(float_of_string
   lxm) }
| ['0'-'9']+ as lxm { INT_LITERAL(int_of_string lxm) }
| '''[^ '\\' ''' '"']?''' as lxm { CHAR_LITERAL(lxm.[1]) }
| ''''\\'['''' '"' '\\' 't' 'n']''' as lxm { CHAR_LITERAL(lxm.[1]) }
| '"' (('\\'['''' '"' '\\' 't' 'n'])+ | [^ '\\' ''' '"']+)* '"' as
   lxm
  { let str = String.sub (lxm) 1 ((String.length lxm) - 2) in
        let unescaped_str = Scanf.unescaped str in
        STRING_LITERAL(unescaped_str) }
| ['A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { SHAPE_ID(lxm) }
| ['a'-'z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped
   char)) }

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

A.2 parser.mly

```
/* Ocamlyacc parser for SOL */

%{
open Ast
%}

%token SEMI LPAREN RPAREN LBRACE RBRACE LSQUARE RSQUARE COMMA
%token PLUS MINUS TIMES DIVIDE MODULO ASSIGN NOT DOT
%token EQ NEQ LT LEQ GT GEQ AND OR
%token RETURN IF WHILE INT FLOAT CHAR STRING FUNC
%token SHAPE CONSTRUCT DRAW /*PARENT EXTENDS MAIN CONSOLEPRINT
   LENGTH SETFRAMERATE */
%token RENDER
/*%token TRANSLATE ROTATE WAIT*/
%token <int> INT_LITERAL
%token <float> FLOAT_LITERAL
%token <char> CHAR_LITERAL
%token <string> STRING_LITERAL
%token <string> ID
%token <string> SHAPE_ID
%token EOF

%right ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE MODULO
%right NOT NEG /* Have to add in parentheses */
%left DOT
%left LPAREN RPAREN LSQUARE RSQUARE

%start program
%type <Ast.program> program

%%

program:
  decls EOF { $1 }

decls:
   /* nothing */ { [], [], [] }
 | decls vdecl { let (v, s, f) = $1 in ($2 :: v), s, f }
 | decls fdecl { let (v, s, f) = $1 in v, s, ($2 :: f) }
```

```
 | decls sdecl { let (v, s, f) = $1 in v, ($2 :: s), f }

fdecl:
   FUNC ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list
      RBRACE /* Handling case for empty return type */
     { { ftype = Void;
   fname = $2;
   formals = $4;
   locals = List.rev $7;
   body = List.rev $8 } }

 | FUNC typ ID LPAREN formals_opt RPAREN LBRACE vdecl_list
    stmt_list RBRACE
     { { ftype = $2;
         fname = $3;
         formals = $5;
         locals = List.rev $8;
         body = List.rev $9 } }

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

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

typ:
    INT { Int }
  | FLOAT { Float }
  | CHAR { Char }
  | STRING { String }
  | SHAPE_ID { Shape($1) }

/*formal_typ:
    typ {$1}
  | formal_typ LSQUARE RSQUARE { Array(0, $1) }*/
/* Removing because we do not need variable length arrays as
   function formal parameters */

local_typ:
    typ {$1}
  | local_typ LSQUARE INT_LITERAL RSQUARE { Array ($3, $1)}
  /* Not adding in Void here*/

vdecl_list:
    /* nothing */    { [] }
```

```
  | vdecl_list vdecl { $2 :: $1 }

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

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

stmt:
    expr SEMI { Expr $1 }
  | RETURN SEMI { Return Noexpr }
  /*| vdecl { VDecl($1, Noexpr) }
  | local_typ ID ASSIGN expr SEMI { VDecl(($1, $2), $4) }*/
  | RETURN expr SEMI { Return $2 }
  | LBRACE stmt_list RBRACE { Block(List.rev $2) }
  | IF LPAREN expr RPAREN stmt { If($3, $5) }
  | WHILE LPAREN expr RPAREN stmt { While($3, $5) }
  | ID DOT RENDER ASSIGN LBRACE stmt_list RBRACE { Shape_render($1,
      List.rev $6) }

/*expr_opt:*/
    /* nothing */ /*{ Noexpr }
  | expr          { $1 }*/
/* Removed because only usage was for FOR statements */

array_expr:
    expr    { [$1] }
  | array_expr COMMA expr { $3 :: $1 }

expr:
    INT_LITERAL          { Int_literal($1) }
  | FLOAT_LITERAL          { Float_literal($1) }
  | CHAR_LITERAL          { Char_literal($1) }
  | STRING_LITERAL          { String_literal($1) }
  | LSQUARE array_expr RSQUARE        { Array_literal(List.length
     $2, List.rev $2) }
  | 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 MODULO 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) }
  | MINUS expr %prec NEG { Unop(Neg, $2) }
  | NOT expr         { Unop(Not, $2) }
  | lvalue ASSIGN expr   { Assign($1, $3) }
  | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
  | SHAPE SHAPE_ID LPAREN actuals_opt RPAREN { Inst_shape($2, $4) }
  | ID DOT ID LPAREN actuals_opt RPAREN { Shape_fn($1, $3, $5) }
  | LPAREN expr RPAREN { $2 }
  | lvalue  { Lval($1) }
  /* TODO: Include expression for typecasting */

lvalue:
    ID    { Id($1) }
  | ID LSQUARE expr RSQUARE     { Access($1, $3) } /*Access a
     specific element of an array*/
  | ID DOT lvalue   { Shape_var($1, $3) }

actuals_opt:
    /* nothing */ { [] }
  | actuals_list  { List.rev $1 }

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

sdecl:
    SHAPE SHAPE_ID LBRACE vdecl_list cdecl ddecl shape_fdecl_list
       RBRACE
      { { sname = $2;
      pname = None;
      member_vs = List.rev $4;
      construct = $5; (* NOTE: Make this optional later *)
      draw = $6;
      member_fs = $7;
      }
    }

cdecl:
   CONSTRUCT LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list
      RBRACE
     { { ftype = Void;
   fname = "constructor";
   formals = $3;
   locals = List.rev $6;
   body = List.rev $7 }
```

```
   }

ddecl:
   DRAW LPAREN RPAREN LBRACE vdecl_list stmt_list RBRACE
     { { ftype = Void;
   fname = "draw";
   formals = [];
   locals = List.rev $5;
   body = List.rev $6 }
   }

shape_fdecl_list:
    /* nothing */ { [] }
  | fdecl_list   { List.rev $1 }

fdecl_list:
    fdecl                    { [$1] }
  | fdecl_list fdecl { $2 :: $1 }
```

A.3 ast.ml

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

type unary_op = Not | Neg

type typ =
          Int
        | Float
        | Char
        | String
        | Void (* For internal use *)
        | Array of int * typ (*first expr is the size of the array
           *)
        | Shape of string
and
   expr =
          Int_literal of int
        | Float_literal of float
        | Char_literal of char
        | String_literal of string
        | Array_literal of int * expr list
        | Binop of expr * op * expr
        | Unop of unary_op * expr
        | Noexpr
        | Assign of lvalue * expr
```

```
        | Call of string * expr list
        | Lval of lvalue
        | Inst_shape of string * expr list
        | Shape_fn of string * string * expr list
and
        lvalue =
          Id of string
        | Access of string * expr
        | Shape_var of string * lvalue

type stmt =
          Block of stmt list
        | Expr of expr
        (* | VDecl of bind * expr *)
        | Return of expr
        | If of expr * stmt
        | While of expr * stmt
        | Shape_render of string * stmt list

type bind = typ * string

type func_dec = {
        fname   :       string;
        ftype   :       typ;
        formals :       bind list;
        locals  :       bind list;
        body    :       stmt list;
}

type shape_dec = {
        sname           :       string;
        pname           :       string option; (*parent name*)
        member_vs       :       bind list;
        construct       :       func_dec;
        draw            :       func_dec;
        member_fs       :       func_dec list;
}

type program = bind list * shape_dec list * func_dec list

(* Pretty-printing functions *)

let string_of_op = function
    Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
```

```
  | Mod -> "%"
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "&&"
  | Or -> "||"

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

let rec string_of_expr = function
    Int_literal(l) -> string_of_int l
  | Float_literal(l) -> string_of_float l
  | Char_literal(l) -> Char.escaped l
  | String_literal(l) -> l
  | Array_literal(len, l) -> string_of_int len ^ ": [" ^ String.
     concat ", " (List.map string_of_expr l) ^ "]"
  | 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(l, e) -> (string_of_lvalue l) ^ " = " ^ string_of_expr e
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ "
         )"
  | Inst_shape(s, el) -> "shape " ^ s ^ "(" ^ String.concat ", " (
     List.map string_of_expr el) ^ ")"
  | Shape_fn(s, f, el) ->
      s ^ "." ^ f ^ "(" ^ String.concat ", " (List.map
         string_of_expr el) ^ ")"
  | Noexpr -> ""
  | Lval(l) -> string_of_lvalue l

 and

string_of_lvalue = function
  Id(s) -> s
| Access(id, idx) -> id ^ "[" ^ string_of_expr idx ^ "]"
| Shape_var(s, v) -> s ^ "." ^ (string_of_lvalue v)

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

```
  | Char -> "char"
  | Void -> "void"
  | String -> "string"
  | Array(l,t) -> string_of_typ t ^ " [" ^ string_of_int l ^ "]"
  | Shape(s) -> "Shape " ^ s

let rec string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "
        }\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  (* | VDecl(id, expr) -> string_of_typ (fst id) ^ " " ^ snd id ^
     ": " ^ string_of_expr expr *)
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
  | If(e, s) -> "if (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt
     s
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^
     string_of_stmt s
  | Shape_render(s, stmts) -> s ^ "." ^ "{\n" ^ String.concat "" (
     List.map string_of_stmt stmts) ^ "}\n"

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

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

let string_of_sdecl sdecl =
  "Shape " ^ sdecl.sname ^ "(" ^ String.concat ", " (List.map snd
     sdecl.construct.formals) ^
  ")\n Member Variables: " ^ String.concat "" (List.map
     string_of_vdecl sdecl.member_vs) ^
  "\n Draw: " ^ string_of_fdecl sdecl.draw ^
  "\n Member functions: " ^ String.concat "" (List.map
     string_of_fdecl sdecl.member_fs)

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

A.4 semant.ml

```
(* Semantic checking for the SOL compiler *)

open Ast
open Sast

module StringMap = Map.Make(String)

type symbol_table = {
  parent: symbol_table option;
  mutable
  variables: bind list
}

type shape_variables = {
  mutable
  num_translates: int;
  shape_type : string;
  shape_inst : string
}

type translation_environment = {
  scope: symbol_table;
  functions: Ast.func_dec StringMap.t;
  shape_vars: shape_variables option;
}

let rec find_variable (scope: symbol_table) name =
  try
    List.find (fun (_, s) -> s = name) scope.variables
  with Not_found ->
    match scope.parent with
    | Some(p) -> find_variable p name
    | _ -> raise Not_found

let find_local (scope: symbol_table) name =
  try
    let _ = List.find (fun (_, s) -> s = name) scope.variables in
      raise(Failure("Local variable already declared with name " ^
        name))
  with Not_found -> ()

(* Semantic checking of a program. Returns void if successful,
   throws an exception if something is wrong.

   Check each global variable, then check each function *)
```

```
let check (globals, shapes, functions) =

  (* Raise an exception if the given list has a duplicate *)
  let report_duplicate exceptf list =
    let rec helper = function
        n1 :: n2 :: _ when n1 = n2 -> raise (Failure (exceptf n1))
      | _ :: t -> helper t
      | [] -> ()
    in helper (List.sort compare list)
  in

  (* Raise an exception if a given binding is to a void type *)
  let check_not_void exceptf = function
      (Void, n) -> raise (Failure (exceptf n))
    | _ -> ()
  in

  (* Raise an exception of the given rvalue type cannot be assigned
      to
     the given lvalue type *)
  let check_assign lvaluet rvaluet err =
    let types = (lvaluet, rvaluet) in match types with
        (Array(l1, t1), Array(l2, t2)) -> if t1 == t2 && l1 == l2
           then lvaluet else raise err
      | (Shape(l_s), Shape(r_s)) -> if l_s = r_s then lvaluet else
         raise err
      | _ -> if lvaluet == rvaluet then lvaluet else raise err
  in

  (* Define global declaration of translate *)
  let translate_fdecl = { ftype = Void; fname = "translate";
     formals = [(Array(2, Int), "disp"); (Int, "t")];
       locals = []; body = [] }
  in

  (**** Checking Global Variables ****)

  List.iter (check_not_void (fun n -> "illegal void global " ^ n))
     globals;

  report_duplicate (fun n -> "duplicate global " ^ n) (List.map snd
      globals);

  (**** Checking Shapes ****)

  report_duplicate (fun n -> "duplicate shape " ^ n)
```

```
  (List.map (fun sd -> sd.sname) shapes);

let shape_decls = List.fold_left (fun m sd -> StringMap.add sd.
   sname sd m)
                       StringMap.empty shapes
in

let shape_decl s = try StringMap.find s shape_decls
     with Not_found -> raise (Failure ("unrecognized shape " ^ s)
        )
in

(**** Checking Functions ****)

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

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

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

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

(* Function declaration for a named function *)
let built_in_decls = StringMap.add "round"
   { ftype = Float; fname = "round"; formals = [(Float, "x")];
     locals = []; body = []} (StringMap.add "cosine"
   { ftype = Float; fname = "cosine"; formals = [(Float, "x")];
     locals = []; body = []} (StringMap.add "sine"
   { ftype = Float; fname = "sine"; formals = [(Float, "x")];
     locals = []; body = []} (StringMap.add "consolePrint"
   { ftype = Void; fname = "consolePrint"; formals = [(String, "x
      ")];
     locals = []; body = [] } (StringMap.add "intToFloat"
   { ftype = Float; fname = "intToFloat"; formals = [(Int, "x")];
     locals = []; body = [] } (StringMap.add "floatToInt"
   { ftype = Int; fname = "floatToInt"; formals = [(Float, "x")];
     locals = []; body = [] } (StringMap.add "intToString"
```

```
    { ftype = String; fname = "intToString"; formals = [(Int, "x")
      ];
      locals = []; body = [] } (StringMap.add "floatToString"
    { ftype = String; fname = "floatToString"; formals = [(Float,
      "x")];
      locals = []; body = [] } (StringMap.add "charToString"
    { ftype = String; fname = "charToString"; formals = [(Char, "x
      ")];
      locals = []; body = [] } (StringMap.add "setFramerate"
    { ftype = Void; fname = "setFramerate"; formals = [(Int, "x")
      ];
      locals = []; body = [] } (StringMap.add "getFramerate"
    { ftype = Int; fname = "getFramerate"; formals = [];
      locals = []; body = [] } (StringMap.add "drawCurve"
    { ftype = Void; fname = "drawCurve"; formals =
        [(Array(2, Int), "x"); (Array(2, Int), "y"); (Array(2, Int
          ), "z"); (Int, "stepsize"); (Array(3, Int), "rgb")];
      locals = []; body = [] } (StringMap.add "drawPoint"
    { ftype = Void; fname = "drawPoint"; formals = [(Array(2, Int)
      , "x"); (Array(3, Int), "rgb")];
      locals = []; body = [] } (StringMap.singleton "print"
    { ftype = Void; fname = "print"; formals = [(Array(2, Int), "x
      "); (String, "text"); (Array(3, Int), "rgb")];
      locals = []; body = [] }))))))))))))))
in

let function_decls = List.fold_left (fun m fd -> StringMap.add fd
   .fname fd m)
                     built_in_decls functions
in

let function_decl s s_map = try StringMap.find s s_map
     with Not_found -> raise (Failure ("unrecognized function " ^
         s ^ " in this scope!"))
in

let _ = function_decl "main" function_decls in (* Ensure "main"
   is defined *)

let check_function g_env func =

  List.iter (check_not_void (fun n -> "illegal void formal " ^ n
     ^
    " in " ^ func.fname)) func.formals;

  report_duplicate (fun n -> "duplicate formal " ^ n ^ " in " ^
     func.fname)
```

```
    (List.map snd func.formals);

  List.iter (check_not_void (fun n -> "illegal void local " ^ n ^
    " in " ^ func.fname)) func.locals;

  report_duplicate (fun n -> "duplicate local " ^ n ^ " in " ^
     func.fname)
    (List.map snd func.locals);

  (* Type of each variable (global, formal, or local *)
  (* let symbols = List.fold_left (fun m (t, n) -> StringMap.add
     n t m)
      StringMap.empty (globals @ func.formals @ func.locals )
  in

  let type_of_identifier s =
    try StringMap.find s symbols
    with Not_found -> raise (Failure ("undeclared identifier " ^
       s))
  in *)
  let map_op tup = match tup with
      (Add, Int) -> IAdd
    | (Sub, Int) -> ISub
    | (Mult, Int) -> IMult
    | (Div, Int) -> IDiv
    | (Equal, Int) -> IEqual
    | (Neq, Int) -> INeq
    | (Less, Int) -> ILess
    | (Leq, Int) -> ILeq
    | (Greater, Int) -> IGreater
    | (Geq, Int) -> IGeq
    | (And, Int) -> IAnd
    | (Or, Int) -> IOr
    | (Mod, Int) -> IMod
    | (Add, Float) -> FAdd
    | (Sub, Float) -> FSub
    | (Mult, Float) -> FMult
    | (Div, Float) -> FDiv
    | (Equal, Float) -> FEqual
    | (Neq, Float) -> FNeq
    | (Less, Float) -> FLess
    | (Leq, Float) -> FLeq
    | (Greater, Float) -> FGreater
    | (Geq, Float) -> FGeq
    | (Mod, Float) -> FMod
    | (_, _) -> raise(Failure("Invalid operation " ^ (
       string_of_op (fst tup)) ^ " for type " ^ (string_of_typ (
```

```
        snd tup)))) in

    (* Return the type of an expression or throw an exception *)
    let rec expr env = function
              Int_literal i -> SInt_literal(i), Int
      | Float_literal f -> SFloat_literal(f), Float
      | Char_literal c -> SChar_literal(c), Char
      | String_literal s -> SString_literal(s), String
      | Array_literal(l, s) as a -> let prim_type = List.fold_left
         (fun t1 e -> let t2 = snd (expr env e) in
          if t1 == t2 then t1
          else raise (Failure("Elements of differing types found in
              array " ^ string_of_expr (a) ^ ": " ^
            string_of_typ t1 ^ ", " ^ string_of_typ t2)))
        (snd (expr env (List.hd (s)))) (List.tl s) in
        (if l == List.length s then
          let s_s = List.map (fun e -> expr env e) s in
          SArray_literal(l, s_s), Array(l, prim_type)
        else raise(Failure("Something wrong with auto-assigning
           length to array literal " ^ string_of_expr a)))
      | Binop(e1, op, e2) as e ->
          let ta = expr env e1 and tb = expr env e2
          in let _, t1 = ta and _, t2 = tb in
                (match op with
            Add | Sub | Mult | Div | Mod when t1 = Int && t2 = Int
               -> SBinop(ta, map_op (op, Int), tb), Int
          | Add | Sub | Mult | Div | Mod when t1 = Float && t2 =
             Float -> SBinop(ta, map_op (op, Float), tb), Float
                | Equal | Neq when t1 = t2 && t1 = Int -> SBinop(ta
                   , map_op (op, Int), tb), Int
          | Equal | Neq when t1 = t2 && t1 = Float -> SBinop(ta,
             map_op (op, Float), tb), Int
                | Less | Leq | Greater | Geq when t1 = Int && t2 =
                   Int -> SBinop(ta, map_op (op, Int), tb), Int
          | Less | Leq | Greater | Geq when t1 = Float && t2 =
             Float -> SBinop(ta, map_op (op, Float), tb), Int
                | And | Or when t1 = Int && t2 = Int -> SBinop(ta,
                   map_op (op, Int), tb), Int
          | _ -> raise (Failure ("illegal binary operator " ^
                      string_of_typ t1 ^ " " ^ string_of_op op ^ "
                         " ^
                      string_of_typ t2 ^ " in " ^ string_of_expr e)
                         )
                )
      | Unop(op, e) as ex ->
          let t1 = expr env e
          in let _, t = t1 in
```

```
      (match op with
        Neg when t = Int -> SUnop(INeg, t1), Int
| Neg when t = Float -> SUnop(FNeg, t1), Float
      | Not when t = Int -> SUnop(INot, t1), Int
| _ -> raise (Failure ("illegal unary operator " ^ string_of_uop
    op ^
                        string_of_typ t ^ " in " ^
                          string_of_expr ex))
      )
    | Noexpr -> SNoexpr, Void
    | Assign(lval, e) as ex ->
        let (slval, lt) = lval_expr env lval and (rexpr, rt) =
           expr env e in
      ignore(check_assign lt rt (Failure ("illegal assignment " ^
          string_of_typ lt ^
          " = " ^ string_of_typ rt ^ " in " ^
          string_of_expr ex)));
      SAssign(slval, (rexpr, rt)), lt
    | Call(fname, actuals) as call -> let fd = function_decl
       fname env.functions in
        ignore(if (fname = "drawCurve" || fname = "drawPoint"||
           fname = "print") then
          if func.fname = "draw"
          then ()
          else raise(Failure("drawCurve/drawPoint/print can only
             be called within a draw()!"))
        else ()
        );
        if List.length actuals != List.length fd.formals then
         raise (Failure ("expecting " ^ string_of_int
           (List.length fd.formals) ^ " arguments in " ^
              string_of_expr call))
        else (* TODO: Add special case for checking type of
           actual array vs formal array *)
          List.iter2 (fun (ft, _) e -> let _, et = expr env e in
            ignore (check_assign ft et
              (Failure ("illegal actual argument found " ^
                 string_of_typ et ^
              " expected " ^ string_of_typ ft ^ " in " ^
                 string_of_expr e))))
           fd.formals actuals;
          let sactuals = List.map (fun a -> expr env a) actuals
             in
          (* Not converting the body to a list of stmt_details,
             to prevent recursive conversions,
           and also because this detail is not needed when making
               a function call *)
```

```
          let s_fd = {sfname = fd.fname; styp = fd.ftype;
             sformals = fd.formals; slocals = fd.locals;
            sbody = []} in
          (* Adding in sequence in which translate is called,
             along with a reference to the shape *)
          let sactuals = (if fname = "translate" then (match env.
             shape_vars with
              Some(v) -> v.num_translates <- v.num_translates +
                 1;
                (SInt_literal(v.num_translates - 1), Int) ::
                  (SLval(SId(v.shape_inst), Shape(v.shape_type)),
                      Shape(v.shape_type)) :: sactuals
            | _ -> raise(Failure("Translate called in non-render
               block!")))
          else (sactuals)) in
          SCall(s_fd, sactuals), fd.ftype
      | Shape_fn(s, fname, actuals) as call -> (try
          let (t, _) = find_variable env.scope s in
          match t with
            Shape(sname) -> let sd = shape_decl sname in
              let fd = try List.find (fun member_fd -> fname =
                 member_fd.fname) sd.member_fs
                with Not_found -> raise(Failure("Member function "
                   ^ fname ^ " not found in shape declaration " ^
                    sname)) in
              if List.length actuals != List.length fd.formals then
                raise (Failure ("expecting " ^ string_of_int
                  (List.length fd.formals) ^ " arguments in " ^
                     string_of_expr call))
               else (* TODO: Add special case for checking type of
                  actual array vs formal array *)
                List.iter2 (fun (ft, _) e -> let _, et = expr env e
                   in
                   ignore (check_assign ft et
                     (Failure ("illegal actual argument found " ^
                        string_of_typ et ^
                     " expected " ^ string_of_typ ft ^ " in " ^
                        string_of_expr e))))
                  fd.formals actuals;
                let sactuals = List.map (fun a -> expr env a)
                   actuals in
                let s_fd = {sfname = fd.fname; styp = fd.ftype;
                   sformals = fd.formals; slocals = fd.locals;
                  sbody = []} in
                  (* Not converting the body to a list of
                     stmt_details, to prevent recursive conversions,
```

```
                and also because this detail is not needed when
                    making a function call *)
              SShape_fn(s, t, s_fd, sactuals), fd.ftype
          | _ -> raise(Failure("Member function access " ^ fname
            ^ " for a non-shape variable " ^ s))
        with Not_found -> raise(Failure("Undeclared identifier "
          ^ s)))
    | Lval l -> let (slval_det, ltyp) = (lval_expr env l) in
      SLval(slval_det), ltyp
    | Inst_shape (sname, actuals) ->
    (* Check if the shape exists *)
      let sd = shape_decl sname in
       if List.length actuals != List.length sd.construct.formals
          then
         raise (Failure ("expecting " ^ string_of_int
           (List.length sd.construct.formals) ^ " arguments in "
             ^ string_of_sdecl sd))
       else (* TODO: Add special case for checking type of actual
          array vs formal array *)
         List.iter2 (fun (ft, _) e -> let _, et = expr env e in
            ignore (check_assign ft et
              (Failure ("illegal actual argument found " ^
                 string_of_typ et ^
              " expected " ^ string_of_typ ft ^ " in " ^
                 string_of_expr e))))
           sd.construct.formals actuals;
        let sactuals = List.map (fun a -> expr env a) actuals in
        let s_sd = {ssname = sd.sname; spname = sd.pname;
           smember_vs = sd.member_vs; sconstruct = {sfname = "
           Construct";
           styp = Void; sformals = []; slocals = []; sbody = []};
              sdraw = {sfname = "Draw";
           styp = Void; sformals = []; slocals = []; sbody = []};
              smember_fs = []} in
           (* Not converting the shape completely, to prevent
              recursive conversions,
           and also because this detail is not needed when making
              a shape instantiation *)
       SInst_shape(s_sd, sactuals), Shape(sname)

and lval_expr env = function
    Id s -> (try
      let (t, _) = find_variable env.scope s in
      ((SId(s), t), t)
      with Not_found -> raise(Failure("Undeclared identifier "
        ^ s)))
  | Access(id, idx) -> (try
```

```
        let (t, _) = find_variable env.scope id
        and (idx', t_ix) = expr env idx in
        let eval_type = function
          Array(_, a_t) -> if t_ix == Int
          (* Note: Cannot check if index is within array bounds
             because the value cannot be evaluated at this stage
             *)
            then a_t
            else raise (Failure("Improper array element access:
               ID " ^ id ^ ", index " ^
              string_of_expr idx))
        | _ -> raise (Failure(id ^ "is not an array type"))
        in ((SAccess(id, (idx', t_ix)), t), eval_type t)
        with Not_found -> raise(Failure("Undeclared identifier "
           ^ id)))
    | Shape_var(s, v) -> try
        let (t, _) = find_variable env.scope s in
        match t with
          Shape(sname) -> let sd = shape_decl sname in
            let shape_scope = {parent = Some(env.scope);
               variables = env.scope.variables @ sd.member_vs}
               in
            let shape_env = {env with scope = shape_scope} in
            let (v_slval, val_typ) = (lval_expr shape_env v) in
            ((SShape_var(s, v_slval), t), val_typ)
            (* (match v_slval with
              SId(v_n), _ -> let (v_t, _) = try List.find (fun
                 (_, n) -> n = v_n) sd.member_vs
                with Not_found -> raise(Failure("Member
                   variable " ^ v_n ^ " not found in shape
                   declaration " ^ sname)) in
              ((SShape_var(s, v_slval), t), val_typ)
            | SAccess(id, _), _ -> let _ = try List.find (fun (
              _, n) -> n = id) sd.member_vs
                with Not_found -> raise(Failure("Member
                   variable " ^ id ^ " not found in shape
                   declaration " ^ sname)) in
                ignore(print_string (string_of_typ val_typ));
              ((SShape_var(s, v_slval), t), val_typ)
            | SShape_var(member_s, _), _ -> let _ = try List.
               find (fun (_, n) -> n = member_s) sd.member_vs
                with Not_found -> raise(Failure("Member
                   variable " ^ member_s ^ " not found in shape
                    declaration " ^ sname)) in
              ((SShape_var(s, v_slval), t), val_typ)
            ) *)
```

```
          | _ -> raise(Failure("Attempted member variable access
              for a non-shape variable " ^ s))
        with Not_found -> raise(Failure("Undeclared identifier "
          ^ s))

and check_bool_expr env e = (let (e', t) = (expr env e) in if t
    != Int (* This is not supposed to be recursive! *)
 then raise (Failure ("expected Int expression (that evaluates
    to 0 or 1) in " ^ string_of_expr e))
 else (e', t))

(* Verify a statement or throw an exception *)
and stmt env = function
           Block sl -> let rec check_block env = function
       [Return _ as s] -> [stmt env s]
     | Return _ :: _ -> raise (Failure "nothing may follow a
         return")
     | s :: ss -> stmt env s :: check_block env ss
     | [] -> []
    in let scope' = {parent = Some(env.scope); variables = []}
    in let env' = {env with scope = scope'}
    in let sl = check_block env' sl in
    ignore (match (env.shape_vars, env'.shape_vars) with
        Some(v), Some(v') -> ignore(v.num_translates <- v'.
          num_translates)
      | _ -> ());
    scope'.variables <- List.rev scope'.variables;
    SBlock(sl)
  | Expr e -> SExpr(expr env e)
  (* | VDecl(b, e) -> let _ = find_local env.scope (snd b) in
      env.scope.variables <- b :: env.scope.variables;
      (* Check that the expression type is compatible with the
         type of the variable
        EXCEPT when the expression is a Noexpr
      *)
      let lt = fst b in
      let e' = expr env e in
      let rt = snd (e') in let _ = (match rt with
      | Void -> lt
      | _ -> check_assign lt rt "Assign" (Failure ("illegal
         assignment " ^ string_of_typ lt ^
          " = " ^ string_of_typ rt ^ " in " ^
          string_of_expr e))) in
      SVDecl(b, e') *)
  | Return e -> let e', t = expr env e in if t = func.ftype
     then SReturn((e', t)) else
```

```
        raise (Failure ("return gives " ^ string_of_typ t ^ "
          expected " ^
                        string_of_typ func.ftype ^ " in " ^
                          string_of_expr e))

    | If(p, b1) -> let e' = check_bool_expr env p in SIf(e', stmt
        env b1)
    | While(p, s) -> let e' = check_bool_expr env p in SWhile(e',
        stmt env s)
    | Shape_render(s, sl) ->
      match func.fname with
          "main" ->
            (try
            let (t, _) = find_variable env.scope s in
            match t with
                Shape(sname) -> let sd = shape_decl sname in
                  let function_decls = List.fold_left (fun m fd
                     -> StringMap.add fd.fname fd m)
                                env.functions sd.member_fs
                  in
                  (* Add in the translate function *)
                  let function_decls = StringMap.add
                     translate_fdecl.fname translate_fdecl
                     function_decls in

                  let shape_scope = {parent = Some(env.scope);
                     variables = env.scope.variables @ sd.
                     member_vs} in
                  let shape_env_vars = {num_translates = 0;
                     shape_type = sname; shape_inst = s} in
                  let shape_env = {scope = shape_scope; functions
                      = function_decls; shape_vars = Some(
                     shape_env_vars)} in
                  let rec check_block env = function
                      [Return _] -> raise (Failure "No return
                         allowed!")
                    | Return _ :: _ -> raise (Failure "No return
                       allowed!")
                    | s :: ss -> stmt env s :: check_block env ss
                    | [] -> []
                  in let sl = List.rev (check_block shape_env (
                     List.rev sl)) in
                  let num_translates = (match shape_env.
                     shape_vars with
                      Some(v) -> v.num_translates
                    | _ -> raise(Failure("Shape lost its
                       environment variables!"))) in
```

```
                    SShape_render(s, sname, num_translates, sl)
                | _ -> raise(Failure("Attempted render definition
                    for a non-shape variable " ^ s))
              with Not_found -> raise(Failure("Undeclared
                  identifier " ^ s)))
        | _ -> raise(Failure("Render blocks can only be set in
            main()!"))
  in

  let l_scope = {parent = Some(g_env.scope); variables = func.
      formals @ func.locals} in
  let l_env = {g_env with scope = l_scope} in

  {sfname = func.fname; styp = func.ftype; sformals = func.
      formals; slocals = func.locals;
    sbody = let sbl = stmt l_env (Block func.body) in
    match sbl with
        SBlock(sl) -> sl
      | _ -> raise(Failure("This isn't supposed to happen!"))}


in

let check_shape g_env shape =

  List.iter (check_not_void (fun n -> "illegal void member
      variable " ^ n ^
    " in " ^ shape.sname)) shape.member_vs;

  report_duplicate (fun n -> "duplicate member variable " ^ n ^ "
      in " ^ shape.sname)
    (List.map snd shape.member_vs);

  report_duplicate (fun n -> "duplicate member function " ^ n)
    (List.map (fun fd -> fd.fname) shape.member_fs);

  let function_decls = List.fold_left (fun m fd -> StringMap.add
      fd.fname fd m)
                       g_env.functions shape.member_fs
  in

  let s_scope = {parent = Some(g_env.scope); variables = g_env.
      scope.variables @ shape.member_vs} in
  let s_env = {scope = s_scope; functions = function_decls;
      shape_vars = None} in
  {ssname = shape.sname; spname = None; smember_vs = shape.
      member_vs;
```

```
    sconstruct = (let s_construct = check_function s_env shape.
       construct in
      let s_construct = {s_construct with sfname = shape.sname ^
         "__construct"} in
      try( let last_s_construct = List.hd (List.rev s_construct.
         sbody) in (match last_s_construct with
          SReturn(_) -> raise(Failure("Constructor cannot have
             return statement for shape " ^ shape.sname))
        | _ -> s_construct)) with Failure "hd" -> s_construct);
    sdraw = (let s_draw = check_function s_env shape.draw in
      let s_draw = {s_draw with sfname = shape.sname ^ "__draw"}
         in
      try( let last_s_draw = List.hd (List.rev s_draw.sbody) in (
         match last_s_draw with
          SReturn(_) -> raise(Failure("Draw function cannot have
             return statement for shape " ^ shape.sname))
        | _ -> s_draw)) with Failure "hd" -> s_draw);
    smember_fs = (List.map (function f -> let s_f =
       check_function s_env f in
      let s_f = {s_f with sfname = shape.sname ^ "__" ^ s_f.
         sfname} in
      match s_f.styp with
    | Void -> s_f
    | _ -> try(let last_s = List.hd (List.rev s_f.sbody) in (
       match last_s with
      | SReturn(_) -> s_f
      | _ -> raise(Failure("Function must have return statement
         of type " ^ string_of_typ s_f.styp))))
      with Failure "hd" -> s_f
    ) shape.member_fs)}

in

(* Check each individual function *)

let g_scope = {parent = None; variables = globals} in
let g_env = {scope = g_scope; functions = function_decls;
   shape_vars = None} in
let s_shapes = List.map (check_shape g_env) shapes in
let s_functions = List.map (function f -> let s_f =
   check_function g_env f in match s_f.styp with
  | Void -> s_f
  | _ -> let last_s = List.hd (List.rev s_f.sbody) in (match
     last_s with
    | SReturn(_) -> s_f
    | _ -> raise(Failure("Function must have return statement of
       type " ^ string_of_typ s_f.styp)))
```

```
    ) functions in
  let rec find_render c_m sstmt =
    (match sstmt with
        SBlock(sl) -> List.fold_left find_render c_m sl
      | SIf(_, sl) -> find_render c_m sl
      | SWhile(_, sl) -> find_render c_m sl
      | SShape_render(_, _, n, _) -> if n > c_m then n else c_m
      | _ -> c_m)
  in
  let find_translates curr_max s_f =
    let b = s_f.sbody in List.fold_left find_render curr_max b
  in
  let max_translates = List.fold_left find_translates 0 s_functions
     in
  (globals,
    s_shapes,
    s_functions,
    max_translates
  )
```

A.5 sast.ml

```
open Ast

type sop = IAdd | ISub | IMult | IDiv | IEqual | INeq | ILess |
   ILeq | IGreater | IGeq | IAnd | IOr | IMod |
            FAdd | FSub | FMult | FDiv | FEqual | FNeq | FLess |
               FLeq | FGreater | FGeq | FMod
(* I = integer, F = floats, may add strings *)

type sunary_op = INot | INeg | FNeg

type sexpr_detail =
      SInt_literal of int
    | SFloat_literal of float
    | SChar_literal of char
    | SString_literal of string
    | SArray_literal of int * sexpr list
    | SBinop of sexpr * sop * sexpr
    | SUnop of sunary_op * sexpr
    | SNoexpr
    | SAssign of slvalue * sexpr
    | SCall of sfunc_dec * sexpr list
    | SLval of slvalue
    | SInst_shape of sshape_dec * sexpr list
    | SShape_fn of string * typ * sfunc_dec * sexpr list
```

```
and
  sexpr = sexpr_detail * typ
and
  slvalue_detail =
      SId of string          (* VDecl ? of bind * expr *)
    | SAccess of string * sexpr
    | SShape_var of string * slvalue
and
  slvalue = slvalue_detail * typ
and
  stmt_detail =
      SBlock of stmt_detail list
    | SExpr of sexpr
    (* | SVDecl of bind * sexpr *)
    | SReturn of sexpr
    | SIf of sexpr * stmt_detail
    | SWhile of sexpr * stmt_detail
    | SShape_render of string * string * int * stmt_detail list
and
  sfunc_dec = {
    sfname    :    string;
    styp    :     typ;
    sformals    :    bind list;
    slocals    :    bind list;
    sbody    :    stmt_detail list;
}
and
  sshape_dec = {
  ssname   : string;
  spname   :   string option; (*parent name*)
  smember_vs : bind list;
  sconstruct : sfunc_dec;
  sdraw    : sfunc_dec;
  smember_fs : sfunc_dec list;
}

type sprogram = bind list * sshape_dec list * sfunc_dec list

(* Pretty-printing functions *)

let string_of_sop = function
    IAdd -> "+"
  | ISub -> "-"
  | IMult -> "*"
  | IDiv -> "/"
  | IMod -> "%"
  | IEqual -> "=="
```

```
  | INeq -> "!="
  | ILess -> "<"
  | ILeq -> "<="
  | IGreater -> ">"
  | IGeq -> ">="
  | IAnd -> "&&"
  | IOr -> "||"
  | FAdd -> "+"
  | FSub -> "-"
  | FMult -> "*"
  | FDiv -> "/"
  | FMod -> "%"
  | FEqual -> "=="
  | FNeq -> "!="
  | FLess -> "<"
  | FLeq -> "<="
  | FGreater -> ">"
  | FGeq -> ">="

let string_of_suop = function
    INeg -> "-"
  | INot -> "!"
  | FNeg -> "-"

let rec string_of_sexpr (s: sexpr) = match fst s with
    SInt_literal(l) -> string_of_int l
  | SFloat_literal(l) -> string_of_float l
  | SChar_literal(l) -> Char.escaped l
  | SString_literal(l) -> l
  | SArray_literal(len, l) -> string_of_int len ^ ": [" ^ String.
     concat ", " (List.map string_of_sexpr l) ^ "]"
  | SBinop(e1, o, e2) ->
      string_of_sexpr e1 ^ " " ^ string_of_sop o ^ " " ^
        string_of_sexpr e2
  | SUnop(o, e) -> string_of_suop o ^ string_of_sexpr e
  | SAssign(l, e) -> (string_of_slvalue l) ^ " = " ^ string_of_sexpr
      e
  | SCall(f, el) ->
      string_of_sfdecl f ^ "(" ^ String.concat ", " (List.map
        string_of_sexpr el) ^ ")"
  | SInst_shape(s, el) -> "shape " ^ s.ssname ^ "(" ^ String.concat
     ", " (List.map string_of_sexpr el) ^ ")"
  | SShape_fn(s, styp, f, el) ->
      s ^ "(" ^ (string_of_typ styp) ^ ")." ^ string_of_sfdecl f ^ "
        (" ^ String.concat ", " (List.map string_of_sexpr el) ^ ")"
  | SNoexpr -> ""
  | SLval(l) -> string_of_slvalue l
```

```
and string_of_slvalue = function
  SId(s), _ -> s
| SAccess(id, idx), _ -> id ^ "[" ^ string_of_sexpr idx ^ "]"
| SShape_var(s, v), _ -> s ^ "." ^ (string_of_slvalue v)

and string_of_sstmt = function
   SBlock(stmts) ->
     "{\n" ^ String.concat "" (List.map string_of_sstmt stmts) ^ "
        }\n"
 | SExpr(expr) -> string_of_sexpr expr ^ ";\n";
 (* | SVDecl(id, expr) -> string_of_typ (fst id) ^ " " ^ snd id ^
    ": " ^ string_of_sexpr expr *)
 | SReturn(expr) -> "return " ^ string_of_sexpr expr ^ ";\n";
 | SIf(e, s) -> "if (" ^ string_of_sexpr e ^ ")\n" ^
    string_of_sstmt s
 | SWhile(e, s) -> "while (" ^ string_of_sexpr e ^ ") " ^
    string_of_sstmt s
 | SShape_render(s, sname, num_t, stmts) -> sname ^ " " ^ s ^ ".
    render(" ^ string_of_int(num_t) ^ "){\n" ^
     String.concat "" (List.map string_of_sstmt stmts) ^ "}\n"

and string_of_svdecl (t, id) = string_of_typ t ^ " " ^ id ^ ";\n"

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

 let string_of_ssdecl sdecl =
  "Shape " ^ sdecl.ssname ^ "(" ^ String.concat ", " (List.map snd
     sdecl.sconstruct.sformals) ^
  ")\n Member Variables: " ^ String.concat "" (List.map
     string_of_svdecl sdecl.smember_vs) ^
  "\n Draw: " ^ string_of_sfdecl sdecl.sdraw ^
  "\n Member functions: " ^ String.concat "" (List.map
     string_of_sfdecl sdecl.smember_fs)

let string_of_sprogram (vars, shapes, funcs, _) =
 String.concat "" (List.map string_of_svdecl vars) ^ "\n" ^
 String.concat "\n" (List.map string_of_ssdecl shapes) ^ "\n" ^
 String.concat "\n" (List.map string_of_sfdecl funcs)
```

A.6 codegen.ml

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

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

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

Detailed documentation on the OCaml LLVM library:

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

*)

module L = Llvm
module A = Ast
module S = Sast

module StringMap = Map.Make(String)

(* Define helper function to find index of an element in a list *)
let rec index_of cmp lst idx = match lst with
| [] -> raise(Failure("Element not found!"))
| hd::tl -> if (cmp hd) then idx else index_of cmp tl (idx + 1)

(* Define helper function to create list of integers in a range *)
let range a b =
  let rec aux a b =
    if a > b then [] else a :: aux (a+1) b  in
  if a > b then List.rev (aux b a) else aux a b;;

let translate (globals, shapes, functions, max_translates) =
  (* ignore(print_string(string_of_int max_translates)); *)
  let context = L.global_context () in
  let the_module = L.create_module context "SOL"
  and i32_t  = L.i32_type   context
  and f32_t  = L.double_type context
  and i8_t   = L.i8_type    context
  and void_t = L.void_type  context in

  (* Create map of shape name to its definition, for convenience *)
  let shape_defs = List.fold_left
    (fun m sshape -> StringMap.add sshape.S.ssname sshape m)
    StringMap.empty shapes in
  let shape_def s = (try StringMap.find s shape_defs
```

```
    with Not_found -> raise(Failure("shape_def Not_found! " ^ s)))
      in

let named_shape_types = List.fold_left
  (fun m ssdecl -> let name = ssdecl.S.ssname in StringMap.add
     name (L.named_struct_type context name) m)
  StringMap.empty shapes in
let shape_type s = (try StringMap.find s named_shape_types
  with Not_found -> raise(Failure("shape_type Not_found!"))) in

let rec ltype_of_typ = function
    A.Int -> i32_t
  | A.Float -> f32_t
  | A.Char -> i8_t
  | A.String -> L.pointer_type i8_t
  | A.Void -> void_t
  | A.Array(l, t) -> L.array_type (ltype_of_typ t) l
  | A.Shape(s) -> shape_type s
  in

(* Declare each global variable; remember its value in a map *)
let global_vars =
  let global_var m (t, n) =
    let init = L.const_int (ltype_of_typ t) 0
    in StringMap.add n (L.define_global n init the_module) m in
  List.fold_left global_var StringMap.empty globals in

let global_vars = StringMap.add "_Running"
  (L.define_global "_Running" (L.const_int (L.i1_type context) 0)
     the_module) global_vars in

(* Helper function to resolve booleans depending on the result
   type *)
let conv_bool pred builder =
  let llty_str = L.string_of_lltype (L.type_of pred) in
  (match llty_str with
      "i32" -> (L.build_icmp L.Icmp.Ne pred (L.const_int i32_t 0)
         "tmp" builder)
    | "i1" -> pred
    | _ -> raise(Failure("Type of predicate is wrong!")))
in

(* Helper function to recursively find all child shapes *)
let rec find_child_objs p_lst p_inst p_sname builder =
  let v_lst = (p_sname, p_inst) :: p_lst in
  (* Look through the shape's member variables, to see if it has
     any other shape members *)
```

```
    let sdef = shape_def p_sname in
    List.fold_left (fun l (v_t, v_n) -> match v_t with
        A.Shape(v_sname) -> (* Find reference to variable shape *)
          let index = index_of (fun (_, member_var) -> v_n =
            member_var) sdef.S.smember_vs 0 in
          let v_inst = L.build_struct_gep p_inst index "tmp"
            builder in
          (v_sname, v_inst) :: (find_child_objs l v_inst v_sname
            builder)
      | A.Array(size, A.Shape(v_sname)) ->
          let index = index_of (fun (_, member_var) -> v_n =
            member_var) sdef.S.smember_vs 0 in
          let v_inst = L.build_struct_gep p_inst index "tmp"
            builder in
          let ind_list = List.rev (List.fold_left
            (fun l i -> (L.build_gep v_inst [| L.const_int i32_t 0
              ; L.const_int i32_t i |] "inst" builder) :: l)
            [] (range 0 (size - 1))) in
          List.fold_left (fun l ind_inst -> find_child_objs l
            ind_inst v_sname builder) l ind_list
      | _ -> l) v_lst sdef.S.smember_vs
  in

  (* Instantiate global constants used for printing/comparisons,
     once *)
  let string_format_str = L.define_global "fmt" (L.const_stringz
     context "%s\n") the_module in
  let int_format_str = L.define_global "int_fmt" (L.const_stringz
     context "%d") the_module in
  let float_format_str = L.define_global "flt_fmt" (L.const_stringz
      context "%f") the_module in
  let char_format_str = L.define_global "char_fmt" (L.const_stringz
      context "%c") the_module in

  (* Declare printf(), which the consolePrint built-in function
     will call *)
  let printf_t = L.var_arg_function_type i32_t [| L.pointer_type
     i8_t |] in
  let printf_func = L.declare_function "printf" printf_t the_module
      in

  (* Declare the built-in startSDL(), which initializes the SDL
     environment *)
  let startSDL_t = L.var_arg_function_type i32_t [| |] in
  let startSDL_func = L.declare_function "startSDL" startSDL_t
     the_module in
```

```
(* Declare the built-in onRenderStartSDL(), which runs before
   every render loop in SDL *)
let onRenderStartSDL_t = L.var_arg_function_type void_t [| |] in
let onRenderStartSDL_func = L.declare_function "onRenderStartSDL"
    onRenderStartSDL_t the_module in

(* Declare the built-in onRenderFinishSDL(), which runs after
   every render loop in SDL *)
let onRenderFinishSDL_t = L.var_arg_function_type void_t [| |] in
let onRenderFinishSDL_func = L.declare_function "
   onRenderFinishSDL" onRenderFinishSDL_t the_module in

(* Declare the built-in stopSDL(), which cleans up and exits the
   SDL environment *)
let stopSDL_t = L.var_arg_function_type i32_t [| |] in
let stopSDL_func = L.declare_function "stopSDL" stopSDL_t
   the_module in

(* Declare trignometric sine function which accepts angle in
   degrees *)
let sine_t = L.var_arg_function_type f32_t [|f32_t|] in
let sine_func = L.declare_function "sine" sine_t the_module in

(* Declare trignometric sine function which accepts angle in
   degrees *)
let cosine_t = L.var_arg_function_type f32_t [|f32_t|] in
let cosine_func = L.declare_function "cosine" cosine_t the_module
    in

(* Declare trignometric sine function which accepts angle in
   degrees *)
let round_t = L.var_arg_function_type f32_t [|f32_t|] in
let round_func = L.declare_function "round" round_t the_module in

(* Declare the built-in intToFloat() function *)
let intToFloat_t = L.function_type f32_t [|i32_t|] in
let intToFloat_func = L.declare_function "intToFloat"
   intToFloat_t the_module in

(* Declare the built-in floatToInt() function *)
let floatToInt_t = L.function_type i32_t [|f32_t|] in
let floatToInt_func = L.declare_function "floatToInt"
   floatToInt_t the_module in

(* Declare the built-in sprintf() function, used by the *ToString
    functions *)
```

```
let sprintf_t = L.var_arg_function_type i32_t [| L.pointer_type
   i8_t; L.pointer_type i8_t |] in
let sprintf_func = L.declare_function "sprintf" sprintf_t
   the_module in

(* Declare the built-in drawCurve(), which draws a curve in SDL
   *)
let drawCurve_t = L.function_type i32_t
  [|  L.pointer_type (L.array_type i32_t 2) ;  L.pointer_type (L.
     array_type i32_t 2) ;
      L.pointer_type (L.array_type i32_t 2) ; i32_t ;  L.
         pointer_type (L.array_type i32_t 3) |] in
let drawCurve_func = L.declare_function "drawCurve" drawCurve_t
   the_module in

(* Declare the built-in drawPoint(), which draws a point in SDL
   *)
let drawPoint_t = L.function_type i32_t
  [| L.pointer_type (L.array_type i32_t 2) ; L.pointer_type (L.
     array_type i32_t 3) |] in
let drawPoint_func = L.declare_function "drawPoint" drawPoint_t
   the_module in

(* Declare the built-in drawPoint(), which draws a point in SDL
   *)
let print_t = L.function_type i32_t
  [| L.pointer_type (L.array_type i32_t 2) ; L.pointer_type i8_t
     ; L.pointer_type (L.array_type i32_t 3) |] in
let print_func = L.declare_function "print" print_t the_module in

(* Declare the built-in setFramerate() function *)
let setFramerate_t = L.function_type void_t [| i32_t |] in
let setFramerate_func = L.declare_function "setFramerate"
   setFramerate_t the_module in

(* Declare the built-in getFramerate() function *)
let getFramerate_t = L.function_type i32_t [| |] in
let getFramerate_func = L.declare_function "getFramerate"
   getFramerate_t the_module in

(* Declare the built-in allocTranslateArray(), which allocates
   space for displacements to shapes*)
let allocTranslateArrayX_t = L.function_type (L.pointer_type
   i32_t)
  [| L.pointer_type i32_t ; L.pointer_type i32_t; i32_t ; L.
     pointer_type i32_t|] in
```

```
let allocTranslateArrayX_func = L.declare_function "
   allocTranslateArrayX" allocTranslateArrayX_t the_module in

(* Declare the built-in allocTranslateArray(), which allocates
   space for displacements to shapes*)
let allocTranslateArrayY_t = L.function_type (L.pointer_type
   i32_t)
  [| L.pointer_type i32_t ; L.pointer_type i32_t; i32_t ; L.
     pointer_type i32_t|] in
let allocTranslateArrayY_func = L.declare_function "
   allocTranslateArrayY" allocTranslateArrayY_t the_module in

(* Declare the built-in translateCurve(), which translates a
   curve by some displacement in SDL *)
let translateCurve_t = L.function_type  void_t
  [|  L.pointer_type (L.array_type i32_t 2) ;  L.pointer_type (L.
     array_type i32_t 2) ;
      L.pointer_type (L.array_type i32_t 2) ; L.pointer_type
         i32_t ; L.pointer_type i32_t ; i32_t ; i32_t |] in
let translateCurve_func = L.declare_function "translateCurve"
   translateCurve_t the_module in

(* Declare the built-in translatePoint(), which translates a
   point by some displacement in SDL *)
let translatePoint_t = L.function_type  void_t
  [|  L.pointer_type (L.array_type i32_t 2) ; L.pointer_type
     i32_t ; L.pointer_type i32_t ; i32_t ; i32_t |] in
let translatePoint_func = L.declare_function "translatePoint"
   translatePoint_t the_module in

(* Define each function (arguments and return type) so we can
   call it *)
let function_decls =
  let function_decl m sfdecl =
    let name = sfdecl.S.sfname
    and formal_types =
      Array.of_list (List.map
        (fun (t,_) -> let ltyp = ltype_of_typ t in
          match t with
            A.Array(_) -> L.pointer_type (ltyp)
          | _ -> ltyp)
      sfdecl.S.sformals)
    in let ftype = (match name with
        "main" -> L.function_type i32_t formal_types
      | _ -> L.function_type (ltype_of_typ sfdecl.S.styp)
         formal_types) in
```

```
    StringMap.add name (L.define_function name ftype the_module,
       sfdecl) m in
  List.fold_left function_decl StringMap.empty functions in

(* Add in member functions for each shape *)
let function_decls =
  let shape_function_decl m ssdecl =
  let sname = ssdecl.S.ssname in
    let m = List.fold_left (fun m smember_f ->
      let f_name = smember_f.S.sfname
      and formal_types =
        Array.of_list (L.pointer_type (shape_type sname) ::
          List.map (fun (t,_) -> let ltyp = ltype_of_typ t in
            match t with
              A.Array(_) -> L.pointer_type (ltyp)
            | _ -> ltyp) smember_f.S.sformals)
      in let ftype = L.function_type (ltype_of_typ smember_f.S.
         styp) formal_types in
      StringMap.add f_name (L.define_function f_name ftype
         the_module, smember_f) m)
    m ssdecl.S.smember_fs in
    (* Add in each constructor and draw as well *)
    let construct_name = ssdecl.S.sconstruct.S.sfname and
    formal_types = Array.of_list (List.map (fun (t,_) -> let ltyp
       = ltype_of_typ t in
         match t with
           A.Array(_) -> L.pointer_type (ltyp)
         | _ -> ltyp)  ssdecl.S.sconstruct.S.sformals) in
    let ftype = L.function_type (L.pointer_type (shape_type sname
       )) formal_types in
    let m = StringMap.add construct_name (L.define_function
       construct_name ftype the_module, ssdecl.S.sconstruct) m in
    let draw_name = ssdecl.S.sdraw.S.sfname and
    formal_types = [| L.pointer_type (shape_type sname) |]
      in let ftype = L.function_type (void_t) formal_types in
    StringMap.add draw_name (L.define_function draw_name ftype
       the_module, ssdecl.S.sdraw) m in
  List.fold_left shape_function_decl function_decls shapes in

(* Fill in the body of each shape type *)
let shape_decl ssdecl =
  let name = ssdecl.S.ssname in
  let s_type = shape_type name in
  let lmember_vs = List.rev (List.fold_left (fun l (t, _) -> (
     ltype_of_typ t) :: l ) [] ssdecl.S.smember_vs) in
  let lmember_fs = List.rev (List.fold_left (fun l smember_f ->
    let formal_types =
```

```
      Array.of_list (List.map (fun (t,_) -> ltype_of_typ t)
        smember_f.S.sformals) in
    let ftype = L.function_type (ltype_of_typ smember_f.S.styp)
      formal_types in
    (L.pointer_type ftype) :: l ) [] ssdecl.S.smember_fs) in
  let lptr_members = [L.array_type i32_t max_translates; L.
    array_type i32_t max_translates;
    L.array_type i32_t max_translates; L.pointer_type i32_t; L.
      pointer_type i32_t; i32_t] in
  (L.struct_set_body s_type (Array.of_list(lmember_vs @
    lmember_fs @ lptr_members)) false) in

ignore(List.iter shape_decl shapes);

(* Fill in the body of the given function *)
let build_function_body sfdecl member_vars =
  (* ignore(print_string (sfdecl.S.sfname ^ "\n")); *)
  let (the_function, _) = (try StringMap.find sfdecl.S.sfname
    function_decls
    with Not_found -> raise(Failure("build_function_body
      Not_found!"))) in
  let builder = L.builder_at_end context (L.entry_block
    the_function) in

  (* SPECIAL CASE: For the main(), add in a call to the
    initalization of the SDL window *)
  let _ = match sfdecl.S.sfname with
      "main" -> ignore(L.build_call startSDL_func [|  |] "
        startSDL" builder)
    | _ -> () in
  (* TODO: Consider storing the returned value somewhere, return
    that as an error *)

  let const_zero = L.const_int i32_t 0 in

  (* Construct the function's "locals": formal arguments and
    locally
    declared variables.  Allocate each on the stack, initialize
      their
    value, if appropriate, and remember their values in the "
      locals" map *)
  let local_vars =
    let add_formal m (t, n) p =
    (* Special case: for member variables, add the first formal (
      pointer to the instance) directly into the map *)
    if (String.length n > 2 && (String.sub n 0 2) = "__")
    then (StringMap.add n p m)
```

```
      else(
        L.set_value_name n p;
          let local =
            (match t with
            (* For arrays, use the pointer directly *)
              A.Array(_) -> p
            | _ -> let l = L.build_alloca (ltype_of_typ t) n
               builder in
                ignore (L.build_store p l builder); l) in
                StringMap.add n local m
      ) in

      let add_local m (t, n) =
        (* Special case: for constructors, don't re-add the local
           into the map *)
        if (String.length n > 2 && (String.sub n 0 2) = "__")
        then m
        else (
                let local_var = L.build_alloca (ltype_of_typ t) n
                   builder       in
          StringMap.add n local_var m
        ) in

      let formals = (List.fold_left2 add_formal StringMap.empty
         sfdecl.S.sformals
           (Array.to_list (L.params the_function)) )
      (* (* The only case where a mismatch occurs is for shape-
         member functions, when the first argument is the shape
      - in this case, ignore the first argument *)
      with Invalid_argument("List.fold_left2") -> List.fold_left2
         add_formal StringMap.empty sfdecl.S.sformals
           (List.tl (Array.to_list (L.params the_function))) *) in
      List.fold_left add_local formals sfdecl.S.slocals in

    (* Return the value for a variable or formal argument *)
    let lookup n = try StringMap.find n local_vars
                   with Not_found -> (try StringMap.find n
                      member_vars
                     with Not_found -> (try StringMap.find n
                        global_vars
                       with Not_found -> raise(Failure("lookup
                          Not_found!"))))
    in

    (* Construct code for an expression; return its value *)
    let rec expr builder loadval = function
              S.SInt_literal(i), _ -> L.const_int i32_t i
```

```
      | S.SFloat_literal(f), _ -> L.const_float f32_t f
      | S.SChar_literal(c), _ -> L.const_int i8_t (Char.code c)
      | S.SString_literal(s), _ -> L.build_global_stringptr s "tmp"
        builder
      | S.SNoexpr, _ -> const_zero
      | S.SArray_literal(_, s), (A.Array(_, prim_typ) as t) ->
          let const_array = L.const_array (ltype_of_typ prim_typ) (
             Array.of_list (List.map (fun e -> expr builder true e)
              s)) in
          if loadval then const_array
          else (let arr_ref = L.build_alloca (ltype_of_typ t) "
             arr_ptr" builder in
            ignore(L.build_store const_array arr_ref builder);
              arr_ref)
      | S.SArray_literal(_, _), _ -> raise(Failure("Invalid Array
       literal being created!"))
      | S.SBinop (e1, op, e2), _ ->
          let e1' = expr builder true e1
          and e2' = expr builder true e2 in
        (match op with
          S.IAnd -> L.build_and
            (conv_bool e1' builder) (conv_bool e2' builder) "tmp"
              builder
        | S.IOr -> L.build_or
            (conv_bool e1' builder) (conv_bool e2' builder) "tmp"
              builder
        | _ ->  (match op with
            S.IAdd    -> L.build_add
          | S.ISub    -> L.build_sub
          | S.IMult   -> L.build_mul
          | S.IDiv    -> L.build_sdiv
          | S.IMod    -> L.build_srem
          | S.IEqual  -> L.build_icmp L.Icmp.Eq
          | S.INeq    -> L.build_icmp L.Icmp.Ne
          | S.ILess   -> L.build_icmp L.Icmp.Slt
          | S.ILeq    -> L.build_icmp L.Icmp.Sle
          | S.IGreater-> L.build_icmp L.Icmp.Sgt
          | S.IGeq    -> L.build_icmp L.Icmp.Sge
          | S.FAdd    -> L.build_fadd
          | S.FSub    -> L.build_fsub
          | S.FMult   -> L.build_fmul
          | S.FDiv    -> L.build_fdiv
          | S.FMod    -> L.build_frem
          | S.FEqual  -> L.build_fcmp L.Fcmp.Oeq
          | S.FNeq    -> L.build_fcmp L.Fcmp.One
          | S.FLess   -> L.build_fcmp L.Fcmp.Olt
          | S.FLeq    -> L.build_fcmp L.Fcmp.Ole
```

```
        | S.FGreater-> L.build_fcmp L.Fcmp.Ogt
        | S.FGeq    -> L.build_fcmp L.Fcmp.Oge
        | _ -> raise(Failure("Found some binary operator that isn
           't handled!"))
        ) e1' e2' "tmp" builder
      )

    | S.SUnop(op, e), _ ->
        let e' = expr builder true e in
          (match op with
            S.INeg    -> L.build_neg e' "tmp" builder
      | S.INot    -> L.build_icmp L.Icmp.Eq (conv_bool (expr
         builder true e) builder) const_zero "tmp" builder
      | S.FNeg    -> L.build_fneg e' "tmp" builder)
    | S.SAssign (lval, s_e), _ -> let e' = expr builder true s_e
       in
                        ignore (L.build_store e' (lval_expr
                          builder lval) builder); e'
    | S.SCall (s_f, act), _ -> let f_name = s_f.S.sfname in
    let actuals = List.rev (List.map
      (fun (s_e, t) ->
        (* Send a pointer to array types instead of the actual
           array *)
        match t with
          A.Array(_) -> expr builder false (s_e, t)
        | _ -> expr builder true (s_e, t))
      (List.rev act)) in (* Why reverse twice? *)

    (match f_name with
        "consolePrint" -> let fmt_str_ptr =
            L.build_in_bounds_gep string_format_str [| const_zero
                ; const_zero |] "tmp" builder in
          L.build_call printf_func (Array.of_list (fmt_str_ptr ::
              actuals)) "printf" builder
      | "intToString" -> let result = L.build_array_alloca i8_t (
         L.const_int i32_t 12) "intToString" builder in
          let int_fmt_ptr =
            L.build_in_bounds_gep int_format_str [| const_zero ;
               const_zero |] "tmp" builder in
          ignore(L.build_call sprintf_func (Array.of_list (result
              :: int_fmt_ptr :: actuals)) "intToStringResult"
             builder);
          result
      | "floatToString" -> let result = L.build_array_alloca i8_t
          (L.const_int i32_t 20) "floatToString" builder in
          let flt_fmt_ptr =
```

```
            L.build_in_bounds_gep float_format_str [| const_zero
               ; const_zero |] "tmp" builder in
          ignore(L.build_call sprintf_func (Array.of_list (result
             :: flt_fmt_ptr :: actuals)) "floatToStringResult"
            builder);
          result
      | "charToString" -> let result = L.build_array_alloca i8_t
         (L.const_int i32_t 2) "charToString" builder in
          let char_fmt_ptr =
            L.build_in_bounds_gep char_format_str [| const_zero ;
               const_zero |] "tmp" builder in
          ignore(L.build_call sprintf_func (Array.of_list (result
             :: char_fmt_ptr :: actuals)) "charToStringResult"
            builder);
          result
      | "drawCurve" ->
          (* Add displacements to the actuals *)
          let disp_final_x = L.build_load (lookup "__disp_x") "
            load_disp_final_x" builder in
          let disp_final_y = L.build_load (lookup "__disp_y") "
            load_disp_final_y" builder in
          let num_frames = L.build_load (lookup "__num_frames") "
            load_num_frames" builder in
          let translateArgs = List.rev (List.tl (List.tl (List.
            rev actuals))) in
          ignore(L.build_call translateCurve_func
            (Array.of_list
              (List.rev(L.const_int i32_t (1) :: num_frames ::
                 disp_final_y :: disp_final_x :: (List.rev
                 translateArgs) )))
            "" builder);
          ignore(L.build_call drawCurve_func (Array.of_list(
            actuals)) "drawCurve_result" builder);
          (* Reverse the displacement *)
          L.build_call translateCurve_func
            (Array.of_list
              (List.rev(L.const_int i32_t (-1) :: num_frames ::
                 disp_final_y :: disp_final_x :: (List.rev
                 translateArgs) )))
            "" builder
      | "drawPoint" ->
      (* Add displacements to the actuals *)
          let disp_final_x = L.build_load (lookup "__disp_x") "
            load_disp_final_x" builder in
          let disp_final_y = L.build_load (lookup "__disp_y") "
            load_disp_final_y" builder in
```

```
            let num_frames = L.build_load (lookup "__num_frames") "
               load_num_frames" builder in
            let translateArgs = List.rev (List.tl (List.rev actuals
               )) in
            ignore(L.build_call translatePoint_func
              (Array.of_list
                (List.rev(L.const_int i32_t (1) :: num_frames ::
                   disp_final_y :: disp_final_x :: translateArgs)))
              "" builder);
            ignore(L.build_call drawPoint_func (Array.of_list(
               actuals)) "drawPoint_result" builder);
            (* Reverse the displacement *)
            L.build_call translatePoint_func
              (Array.of_list
                (List.rev(L.const_int i32_t (-1) :: num_frames ::
                   disp_final_y :: disp_final_x :: translateArgs)))
              "" builder
        | "print" ->
            let disp_final_x = L.build_load (lookup "__disp_x") "
               load_disp_final_x" builder in
            let disp_final_y = L.build_load (lookup "__disp_y") "
               load_disp_final_y" builder in
            let num_frames = L.build_load (lookup "__num_frames") "
               load_num_frames" builder in
            let translateArgs = List.rev (List.tl (List.tl (List.
               rev actuals))) in
            ignore(L.build_call translatePoint_func
              (Array.of_list
                (List.rev(L.const_int i32_t (1) :: num_frames ::
                   disp_final_y :: disp_final_x :: (List.rev(
                   translateArgs)))))
              "" builder);
            ignore(L.build_call print_func (Array.of_list(actuals))
                "print_result" builder);
            (* Reverse the displacement *)
            L.build_call translatePoint_func
              (Array.of_list
                (List.rev(L.const_int i32_t (-1) :: num_frames ::
                   disp_final_y :: disp_final_x :: (List.rev(
                   translateArgs)))))
              "" builder;
        | "setFramerate" -> L.build_call setFramerate_func (Array.
           of_list(actuals)) "" builder
        | "getFramerate" -> L.build_call getFramerate_func (Array.
           of_list(actuals)) "getFramerate_result" builder
        | "intToFloat" -> L.build_call intToFloat_func (Array.
           of_list(actuals)) "intToFloat_func" builder
```

```
        | "floatToInt" -> L.build_call floatToInt_func (Array.
           of_list(actuals)) "floatToInt_func" builder
        | "sine" -> L.build_call sine_func (Array.of_list(actuals))
            "sine_func" builder
        | "cosine" -> L.build_call cosine_func (Array.of_list(
           actuals)) "cosine_func" builder
        | "round" -> L.build_call round_func (Array.of_list(actuals
           )) "round_func" builder
        | "translate" ->
            (* Extract the first actual - this is the sequence
               number with which to add to the list of
               displacements *)
            let seq = (match List.hd act with
                (S.SInt_literal(_) , _ )as i -> expr builder true i
              | _ -> raise(Failure("Incorrect first argument for
                 translate!"))) in
            let act = List.tl act in
            let actuals = List.tl actuals in
            (* Extract the shape instance *)
            let (main_shape_inst, main_sname) = (match List.hd act
               with
                S.SLval(S.SId(n), _), A.Shape(sname) -> (lookup n,
                   sname)
              | _ -> raise(Failure("Incorrect first argument for
                 translate!"))) in
            (* let act = List.tl act in *)
            let actuals = List.tl actuals in
            let child_shapes = find_child_objs [] main_shape_inst
               main_sname builder in
            ignore(List.iter (fun (inst_sname, shape_inst) ->
              let sdecl = shape_def inst_sname in
              let start_index = (List.length sdecl.S.smember_vs) +
                 (List.length sdecl.S.smember_fs) in
              (* Get access to the correct elements to add in
                 displacement and time values *)
              let disp_ref_x = L.build_gep
                (L.build_struct_gep shape_inst start_index "
                   disp_ref_x" builder)
                [| const_zero ; seq |] "disp_x_seq_ref" builder in
              let disp_ref_y = L.build_gep
                (L.build_struct_gep shape_inst (start_index + 1) "
                   disp_ref_y" builder)
                [| const_zero ; seq |] "disp_y_seq_ref" builder in
              let time_ref = L.build_gep
                (L.build_struct_gep shape_inst (start_index + 2) "
                   time_ref" builder)
```

```
            [| const_zero ; seq |] "disp_time_ref" builder in
               (* const_zero *)
          (* Store values *)
          ignore(L.build_store (L.build_load
            (L.build_gep (List.hd actuals) [| const_zero ;
               const_zero |] "disp_seq_ref" builder)
            "disp_val_x" builder ) disp_ref_x builder);
          ignore(L.build_store (L.build_load
            (L.build_gep (List.hd actuals) [| const_zero ; L.
               const_int i32_t 1 |] "disp_seq_ref" builder)
            "disp_val_y" builder ) disp_ref_y builder);
          ignore(L.build_store (List.hd (List.rev actuals))
             time_ref builder);
        ) child_shapes); const_zero
    | _ -> let (fdef, fdecl), actuals =
        (try StringMap.find f_name function_decls, actuals
        with Not_found ->
        (* In this case, another member function is being
           called from inside a member function *)
        let (sname, inst_name) = (match List.hd sfdecl.S.
           sformals with
            (A.Shape(s), n) -> (s, n)
          | _ -> (* In this case, the member function is being
             called from the constructor *)
              match List.hd sfdecl.S.slocals with
                  (A.Shape(s), n) -> (s, n)
                | _ -> raise(Failure("SCall Not_found"))
        )
        in
        let new_f_name = sname ^ "__" ^ f_name in
        (StringMap.find new_f_name function_decls, (lookup
           inst_name) :: actuals)) in
            let result = (match fdecl.S.styp with A.Void -> ""
                                              | _ -> f_name ^ "
                                                 _result") in
      L.build_call fdef (Array.of_list actuals) result builder)
  | S.SShape_fn(s, styp, s_f, act), _ -> let obj = (lookup s)
     in
      let f_name = (match styp with
          A.Shape(sname) -> sname
        | _ -> raise(Failure("Non-shape type object in member
           function call!"))) ^ "__" ^ s_f.S.sfname in
      let actuals = List.rev (List.map
        (fun (s_e, t) ->
          (* Send a pointer to array types instead of the
             actual array *)
          match t with
```

```
              A.Array(_) -> expr builder false (s_e, t)
            | _ -> expr builder true (s_e, t))
            (List.rev act)) in
        let (fdef, fdecl) = (try StringMap.find f_name
           function_decls with Not_found -> raise(Failure("
           SShape_fn Not_found!"))) in
        let result = (match fdecl.S.styp with A.Void -> ""
                                            | _ -> f_name ^ "
                                               _result") in
        L.build_call fdef (Array.of_list (obj :: actuals)) result
            builder
    | S.SLval(l), _ -> let lval = lval_expr builder l in
        if loadval then L.build_load lval "tmp" builder
        else lval
    | S.SInst_shape(_, sactuals), A.Shape(sname) ->
        let actuals =
        List.rev (List.map (fun (s_e, t) -> let ll_expr = expr
           builder true (s_e, t) in
          (* Send a pointer to array types instead of the actual
             array *)
          match t with
            A.Array(_) -> let copy = L.build_alloca (ltype_of_typ
                t) "arr_copy" builder in
              ignore(L.build_store ll_expr copy builder); copy
          | _ -> ll_expr)
          (List.rev sactuals)) in
        (* Call the constructor *)
        let (constr, _) = (try StringMap.find (sname ^ "
           __construct") function_decls with Not_found -> raise(
           Failure("SInst_shape Not_found!"))) in
        let new_inst = L.build_call constr (Array.of_list actuals
           ) (sname ^ "_inst_ptr") builder in
        L.build_load new_inst (sname ^ "_inst") builder
    | S.SInst_shape(_, _), _ -> raise(Failure("Cannot instantiate
       a shape of non-shape type!"))


  and lval_expr builder = function
    S.SId(s), _ -> lookup s
  | S.SAccess(id, idx), _(* el_typ *) ->
      (* ignore(print_string "access"); *)
      let arr = lookup id in
      let idx' = expr builder true idx in
      (* let arr_len = L.array_length (ltype_of_typ el_typ) in
      if (idx' < const_zero || idx' >= (L.const_int i32_t arr_len
         )) 
```

```
        then raise(Failure("Attempted access out of array bounds
           "))
        (* TODO: figure out how to check for access out of array
           bounds *)
        else  *)L.build_gep arr [| const_zero ; idx' |] "tmp"
           builder
      (*let id' = lookup id
      and idx' = expr builder idx in
      if idx' < (expr builder (A.Int_literal 0)) || idx' > id
         '.(1) then raise(Failure("Attempted access out of array
         bounds"))
      else L.const_int i32_t idx'*)
  | S.SShape_var(s, v), s_t ->
      let rec resolve_shape_var obj var obj_type =
        (* Find index of variable in the shape definition *)
        match obj_type with
            A.Shape(sname) -> let sdef = shape_def sname in
            (match var with
                S.SId(v_n), _ -> let index = index_of (fun (_,
                   member_var) -> v_n = member_var) sdef.S.
                   smember_vs 0 in
                  L.build_struct_gep obj index "tmp" builder
              | S.SAccess(v_n, idx), _ -> let index = index_of (
                 fun (_, member_var) -> v_n = member_var) sdef.S.
                 smember_vs 0 in
                  let arr = L.build_struct_gep obj index "tmp"
                     builder in
                  let idx' = expr builder true idx in
                  L.build_gep arr [| const_zero ; idx' |] "tmp"
                     builder
              | S.SShape_var(member_n, member_v), member_t ->
                  let index = index_of (fun (_, member_var) ->
                     member_n = member_var) sdef.S.smember_vs 0
                     in
                  let id = L.build_struct_gep obj index "tmp"
                     builder in
                  resolve_shape_var id member_v member_t
            )
          | _ -> raise(Failure("Cannot access a shape variable of
             a non-shape type object!"))
      in
      resolve_shape_var (lookup s) v s_t

  in

  (* Invoke "f builder" if the current block doesn't already
     have a terminal (e.g., a branch). *)
```

```
    let add_terminal builder f =
      match L.block_terminator (L.insertion_block builder) with
            Some _ -> ()
      | None -> (* ignore(print_string "Found no return statement
         !");  *)ignore (f builder) in

    (* Build the code for the given statement; return the builder
       for
       the statement's successor *)
    let rec stmt builder = function
            S.SBlock sl -> List.fold_left stmt builder sl
      | S.SExpr e -> ignore (expr builder true e); builder
      (* | S.SVDecl ((t, n), e) -> let var = L.build_alloca (
         ltype_of_typ t) n builder in
          let e' = expr builder e in
          ignore(L.build_store e' var builder); builder *)
      | S.SReturn e -> ignore (match sfdecl.S.styp with
              A.Void -> L.build_ret_void builder
            | _ -> L.build_ret (expr builder true e) builder);
             builder
      | S.SIf (predicate, then_stmt) ->
          let bool_val = conv_bool (expr builder true predicate)
             builder in

          let merge_bb = L.append_block context "merge"
             the_function in

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

          ignore (L.build_cond_br bool_val then_bb merge_bb builder
             );
          L.builder_at_end context merge_bb

      | S.SWhile (predicate, body) ->
          let pred_bb = L.append_block context "while" the_function
              in
          ignore (L.build_br pred_bb builder);

          let body_bb = L.append_block context "while_body"
             the_function in
          add_terminal (stmt (L.builder_at_end context body_bb)
             body)
            (L.build_br pred_bb);
```

```
        let pred_builder = L.builder_at_end context pred_bb in
        let bool_val = conv_bool (expr pred_builder true
           predicate) builder in

        let merge_bb = L.append_block context "merge"
           the_function in
        ignore (L.build_cond_br bool_val body_bb merge_bb
           pred_builder);
        L.builder_at_end context merge_bb
    | S.SShape_render(s, sname, num_t, sl) -> let main_shape_inst
       = lookup s in
        let child_shapes = find_child_objs [] main_shape_inst
           sname builder in
        let builder = List.fold_left stmt builder sl in
        (* For each child shape *)
        ignore(List.iter (fun (inst_sname, shape_inst) ->
          (* Call the external functions for allocTranslateArray
             to get references to the displacements to perform,
             per frame *)
          (* First get references to the correct data within the
             instance *)
          let sdecl = shape_def inst_sname in
          let start_index = (List.length sdecl.S.smember_vs) + (
             List.length sdecl.S.smember_fs) in
          (* ignore(print_string(inst_sname ^ " " ^ (
             string_of_int start_index) ^ "\n")); *)
          let disp_ref_x = L.build_gep
            (L.build_struct_gep shape_inst start_index "
               disp_ref_x" builder)
            [| const_zero ; const_zero|] "fst_disp_ref_x" builder
                in
          let disp_ref_y = L.build_gep
            (L.build_struct_gep shape_inst (start_index + 1) "
               disp_ref_y" builder)
            [| const_zero ; const_zero|] "fst_disp_ref_y" builder
                in
          let time_ref = L.build_gep
            (L.build_struct_gep shape_inst (start_index + 2) "
               time_ref" builder)
            [| const_zero ; const_zero|] "fst_time_ref" builder
               in
          let disp_final_x_ref = (L.build_struct_gep shape_inst (
             start_index + 3) "disp_final_x" builder) in
          let disp_final_y_ref = (L.build_struct_gep shape_inst (
             start_index + 4) "disp_final_y" builder) in
```

```
          let num_frames = (L.build_struct_gep shape_inst (
             start_index + 5) "num_frames_ref" builder) in
          ignore(L.build_store (L.build_call
             allocTranslateArrayX_func
            [| disp_ref_x ; time_ref ; L.const_int i32_t num_t ;
               num_frames |]
            "allocX_result" builder) disp_final_x_ref builder);
          ignore(L.build_store (L.build_call
             allocTranslateArrayY_func
            [| disp_ref_y ; time_ref ; L.const_int i32_t num_t ;
               num_frames |]
            "allocY_result" builder) disp_final_y_ref builder);
          (* let int_fmt_ptr =
              L.build_in_bounds_gep int_format_str [| const_zero
                 ; const_zero |] "tmp" builder in
          ignore(L.build_call printf_func [| int_fmt_ptr ; L.
             build_load (num_frames) "num_frames_loaded" builder
             |] "" builder); *)
        ) child_shapes);
        builder

  in

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

  (* SPECIAL CASE: For the main(), add in a call to the main
     rendering of the SDL window, return its result *)
  let _ = match sfdecl.S.sfname with
      "main" ->
      (* Find all shape objects within scope *)
      let final_objs = List.rev (List.fold_left (fun lst (t, n)
         -> match t with
          A.Shape(sname) -> let inst = lookup n in
            find_child_objs lst inst sname builder
        | A.Array(size, A.Shape(sname)) -> let arr_inst = lookup
           n in
            let ind_list = List.rev (List.fold_left
              (fun l i -> (L.build_gep arr_inst [| const_zero ; L
                 .const_int i32_t i |] "inst" builder) :: l)
              [] (range 0 (size - 1))) in
            List.fold_left (fun l ind_inst -> find_child_objs l
               ind_inst sname builder) lst ind_list
        | _ -> lst)
      [] (List.rev sfdecl.S.slocals)) in

      (* Create a loop while program is running *)
```

```
        let pred_bb = L.append_block context "while" the_function
           in
        ignore (L.build_br pred_bb builder);

        let body_bb = L.append_block context "while_body"
           the_function in
        let builder_body = L.builder_at_end context body_bb in
        (* Build call to onRenderStartSDL() *)
        ignore(L.build_call onRenderStartSDL_func [|  |] ""
           builder_body);

        (* Build a call to each shape's draw function *)
        List.iter (fun (n, o) ->
          let (draw_fn, _) = StringMap.find (n ^ "__draw")
             function_decls in
          ignore(L.build_call draw_fn [| o |] "" builder_body) )
        final_objs;

        (* Build call to onRenderFinishSDL() *)
        ignore(L.build_call onRenderFinishSDL_func [|  |] ""
           builder_body);
        ignore(L.build_br pred_bb builder_body);

        let pred_builder = L.builder_at_end context pred_bb in
        let merge_bb = L.append_block context "merge" the_function
           in
        ignore (L.build_cond_br
          (L.build_load (StringMap.find "_Running" global_vars) "
             _Running_val" pred_builder)
        body_bb merge_bb pred_builder);
        let builder = L.builder_at_end context merge_bb in

        (* (* Free all allocated memory *)
        List.iter (fun (n, o) ->
          let sdecl = shape_def n in
          let start_index = (List.length sdecl.S.smember_vs) + (
             List.length sdecl.S.smember_fs) in
          let disp_x_ref = L.build_load (L.build_struct_gep o (
             start_index + 3) "disp_x" builder) "disp_x_load"
             builder in
          let disp_y_ref = L.build_load (L.build_struct_gep o (
             start_index + 4) "disp_y" builder) "disp_y_load"
             builder in
          ignore(L.build_free disp_x_ref builder);
          ignore(L.build_free disp_y_ref builder);
          ) final_objs; *)
```

```
        let stopSDL_ret = L.build_alloca i32_t "stopSDL_ret"
            builder in
          ignore(L.build_store (L.build_call stopSDL_func [|  |] "
              stopSDL_ret" builder) stopSDL_ret builder);
          ignore(L.build_ret (L.build_load stopSDL_ret "stopSDL_ret
              " builder) builder)
      | _ -> () in

    (* Add a return if the last block falls off the end *)
    (* add_terminal builder (match sfdecl.S.styp with
        A.Void -> L.build_ret_void
      | _ -> L.build_ret const_zero(* L.build_ret (L.const_int (
          ltype_of_typ t) 0) *)) *)
    match sfdecl.S.styp with
        A.Void -> add_terminal builder L.build_ret_void
      | _ -> ()
in

let build_object_function_body sfdecl sdecl =
    let sname = sdecl.S.ssname in
    let stype = shape_type sname in
    let (the_function, _) = (try StringMap.find sfdecl.S.sfname
        function_decls
      with Not_found -> raise(Failure("build_object_function_body
          Not_found!"))) in
    let builder = L.builder_at_end context (L.entry_block
        the_function) in
    let construct_name = sname ^ "__construct" in
    let inst_name = "__" ^ sname ^ "_inst" in

    let shape_inst =
      if sfdecl.S.sfname = construct_name
      (* SPECIAL CASE: For the construct(), add creation of an
          object of the required type *)
      then let inst = L.build_alloca stype inst_name builder in
        let maxFrameIndex = (List.length sdecl.S.smember_vs) + (
            List.length sdecl.S.smember_fs) + 5 in
        ignore(L.build_store (L.const_int i32_t 0) (L.
            build_struct_gep inst maxFrameIndex "num_frames" builder
            ) builder); inst
        (* In all other cases, return the first argument of the
            function *)
      else let inst = Array.get (L.params the_function) 0 in ignore
          (L.set_value_name inst_name inst); inst
    in

    let sfdecl =
```

```
    if sfdecl.S.sfname = construct_name
    then {sfdecl with S.styp = A.Shape(sname); S.slocals = (A.
       Shape(sname), inst_name) :: sfdecl.S.slocals}
    else {sfdecl with S.sformals = (A.Shape(sname), inst_name) ::
        sfdecl.S.sformals}
  in

  (* Create pointers to all member variables *)
  let member_vars = List.fold_left
    (fun m ((_, n), i) -> let member_val = L.build_struct_gep
       shape_inst i n builder in
      StringMap.add n member_val m)
    StringMap.empty (List.mapi (fun i v -> (v, i)) sdecl.S.
       smember_vs) in
  let start_index = (List.length sdecl.S.smember_vs) + (List.
     length sdecl.S.smember_fs) in
  (* Add allocations for pointer members *)
  let member_vars = StringMap.add "__disp_vals_x" (L.
     build_struct_gep shape_inst start_index "disp_vals_x"
     builder) member_vars in
  let member_vars = StringMap.add "__disp_vals_y" (L.
     build_struct_gep shape_inst (start_index + 1) "disp_vals_y"
     builder) member_vars in
  let member_vars = StringMap.add "__time_vals" (L.
     build_struct_gep shape_inst (start_index + 2) "time_vals"
     builder) member_vars in
  let member_vars = StringMap.add "__disp_x" (L.build_struct_gep
     shape_inst (start_index + 3) "disp_x" builder) member_vars
     in
  let member_vars = StringMap.add "__disp_y" (L.build_struct_gep
     shape_inst (start_index + 4) "disp_y" builder) member_vars
     in
  let member_vars = StringMap.add "__num_frames" (L.
     build_struct_gep shape_inst (start_index + 5) "num_frames"
     builder) member_vars in
  let member_vars = if sfdecl.S.sfname = construct_name
    then StringMap.add inst_name shape_inst member_vars
    else member_vars
  in

  (* Build rest of the function body *)
  build_function_body sfdecl member_vars;

  (* SPECIAL CASE: For the construct(), return the instantiated
     object *)
  if sfdecl.S.sfname = construct_name
  then let bbs = L.basic_blocks the_function in
```

```
      let builder = L.builder_at_end context (Array.get bbs ((Array
         .length bbs) - 1)) in
      (* build_function_body would have inserted a void return
         statement at the end; remove this *)
      match L.block_terminator (L.insertion_block builder) with
        Some ins -> (L.delete_instruction ins);
          ignore(L.build_ret shape_inst builder)
      | None -> ignore(L.build_ret shape_inst builder)
    else ()

  in

  List.iter (fun f -> build_function_body f StringMap.empty)
     functions;
  List.iter (fun s ->
    build_object_function_body s.S.sconstruct s;
    build_object_function_body s.S.sdraw s;
    List.iter (fun f -> build_object_function_body f s) s.S.
       smember_fs;)
  shapes;
  the_module
```

A.7 sol.ml

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

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

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

Detailed documentation on the OCaml LLVM library:

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

*)

module L = Llvm
module A = Ast
module S = Sast

module StringMap = Map.Make(String)

(* Define helper function to find index of an element in a list *)
let rec index_of cmp lst idx = match lst with
```

```
| [] -> raise(Failure("Element not found!"))
| hd::tl -> if (cmp hd) then idx else index_of cmp tl (idx + 1)

(* Define helper function to create list of integers in a range *)
let range a b =
  let rec aux a b =
    if a > b then [] else a :: aux (a+1) b  in
  if a > b then List.rev (aux b a) else aux a b;;

let translate (globals, shapes, functions, max_translates) =
  (* ignore(print_string(string_of_int max_translates)); *)
  let context = L.global_context () in
  let the_module = L.create_module context "SOL"
  and i32_t  = L.i32_type   context
  and f32_t  = L.double_type context
  and i8_t   = L.i8_type    context
  and void_t = L.void_type  context in

  (* Create map of shape name to its definition, for convenience *)
  let shape_defs = List.fold_left
    (fun m sshape -> StringMap.add sshape.S.ssname sshape m)
    StringMap.empty shapes in
  let shape_def s = (try StringMap.find s shape_defs
    with Not_found -> raise(Failure("shape_def Not_found! " ^ s)))
       in

  let named_shape_types = List.fold_left
    (fun m ssdecl -> let name = ssdecl.S.ssname in StringMap.add
       name (L.named_struct_type context name) m)
    StringMap.empty shapes in
  let shape_type s = (try StringMap.find s named_shape_types
    with Not_found -> raise(Failure("shape_type Not_found!"))) in

  let rec ltype_of_typ = function
      A.Int -> i32_t
    | A.Float -> f32_t
    | A.Char -> i8_t
    | A.String -> L.pointer_type i8_t
    | A.Void -> void_t
    | A.Array(l, t) -> L.array_type (ltype_of_typ t) l
    | A.Shape(s) -> shape_type s
    in

  (* Declare each global variable; remember its value in a map *)
  let global_vars =
    let global_var m (t, n) =
      let init = L.const_int (ltype_of_typ t) 0
```

```
    in StringMap.add n (L.define_global n init the_module) m in
  List.fold_left global_var StringMap.empty globals in

let global_vars = StringMap.add "_Running"
  (L.define_global "_Running" (L.const_int (L.i1_type context) 0)
     the_module) global_vars in

(* Helper function to resolve booleans depending on the result
   type *)
let conv_bool pred builder =
  let llty_str = L.string_of_lltype (L.type_of pred) in
  (match llty_str with
      "i32" -> (L.build_icmp L.Icmp.Ne pred (L.const_int i32_t 0)
         "tmp" builder)
    | "i1" -> pred
    | _ -> raise(Failure("Type of predicate is wrong!")))
in

(* Helper function to recursively find all child shapes *)
let rec find_child_objs p_lst p_inst p_sname builder =
  let v_lst = (p_sname, p_inst) :: p_lst in
  (* Look through the shape's member variables, to see if it has
     any other shape members *)
  let sdef = shape_def p_sname in
  List.fold_left (fun l (v_t, v_n) -> match v_t with
      A.Shape(v_sname) -> (* Find reference to variable shape *)
        let index = index_of (fun (_, member_var) -> v_n =
           member_var) sdef.S.smember_vs 0 in
        let v_inst = L.build_struct_gep p_inst index "tmp"
           builder in
        (v_sname, v_inst) :: (find_child_objs l v_inst v_sname
           builder)
    | A.Array(size, A.Shape(v_sname)) ->
        let index = index_of (fun (_, member_var) -> v_n =
           member_var) sdef.S.smember_vs 0 in
        let v_inst = L.build_struct_gep p_inst index "tmp"
           builder in
        let ind_list = List.rev (List.fold_left
          (fun l i -> (L.build_gep v_inst [| L.const_int i32_t 0
             ; L.const_int i32_t i |] "inst" builder) :: l)
          [] (range 0 (size - 1))) in
        List.fold_left (fun l ind_inst -> find_child_objs l
           ind_inst v_sname builder) l ind_list
    | _ -> l) v_lst sdef.S.smember_vs
in
```

```
(* Instantiate global constants used for printing/comparisons,
   once *)
let string_format_str = L.define_global "fmt" (L.const_stringz
   context "%s\n") the_module in
let int_format_str = L.define_global "int_fmt" (L.const_stringz
   context "%d") the_module in
let float_format_str = L.define_global "flt_fmt" (L.const_stringz
    context "%f") the_module in
let char_format_str = L.define_global "char_fmt" (L.const_stringz
    context "%c") the_module in

(* Declare printf(), which the consolePrint built-in function
   will call *)
let printf_t = L.var_arg_function_type i32_t [| L.pointer_type
   i8_t |] in
let printf_func = L.declare_function "printf" printf_t the_module
    in

(* Declare the built-in startSDL(), which initializes the SDL
   environment *)
let startSDL_t = L.var_arg_function_type i32_t [| |] in
let startSDL_func = L.declare_function "startSDL" startSDL_t
   the_module in

(* Declare the built-in onRenderStartSDL(), which runs before
   every render loop in SDL *)
let onRenderStartSDL_t = L.var_arg_function_type void_t [| |] in
let onRenderStartSDL_func = L.declare_function "onRenderStartSDL"
    onRenderStartSDL_t the_module in

(* Declare the built-in onRenderFinishSDL(), which runs after
   every render loop in SDL *)
let onRenderFinishSDL_t = L.var_arg_function_type void_t [| |] in
let onRenderFinishSDL_func = L.declare_function "
   onRenderFinishSDL" onRenderFinishSDL_t the_module in

(* Declare the built-in stopSDL(), which cleans up and exits the
   SDL environment *)
let stopSDL_t = L.var_arg_function_type i32_t [| |] in
let stopSDL_func = L.declare_function "stopSDL" stopSDL_t
   the_module in

(* Declare trignometric sine function which accepts angle in
   degrees *)
let sine_t = L.var_arg_function_type f32_t [|f32_t|] in
let sine_func = L.declare_function "sine" sine_t the_module in
```

```
(* Declare trignometric sine function which accepts angle in
   degrees *)
let cosine_t = L.var_arg_function_type f32_t [|f32_t|] in
let cosine_func = L.declare_function "cosine" cosine_t the_module
    in

(* Declare trignometric sine function which accepts angle in
   degrees *)
let round_t = L.var_arg_function_type f32_t [|f32_t|] in
let round_func = L.declare_function "round" round_t the_module in

(* Declare the built-in intToFloat() function *)
let intToFloat_t = L.function_type f32_t [|i32_t|] in
let intToFloat_func = L.declare_function "intToFloat"
   intToFloat_t the_module in

(* Declare the built-in floatToInt() function *)
let floatToInt_t = L.function_type i32_t [|f32_t|] in
let floatToInt_func = L.declare_function "floatToInt"
   floatToInt_t the_module in

(* Declare the built-in sprintf() function, used by the *ToString
    functions *)
let sprintf_t = L.var_arg_function_type i32_t [| L.pointer_type
   i8_t; L.pointer_type i8_t |] in
let sprintf_func = L.declare_function "sprintf" sprintf_t
   the_module in

(* Declare the built-in drawCurve(), which draws a curve in SDL
   *)
let drawCurve_t = L.function_type i32_t
  [|  L.pointer_type (L.array_type i32_t 2) ;  L.pointer_type (L.
     array_type i32_t 2) ;
      L.pointer_type (L.array_type i32_t 2) ; i32_t ;  L.
         pointer_type (L.array_type i32_t 3) |] in
let drawCurve_func = L.declare_function "drawCurve" drawCurve_t
   the_module in

(* Declare the built-in drawPoint(), which draws a point in SDL
   *)
let drawPoint_t = L.function_type i32_t
  [| L.pointer_type (L.array_type i32_t 2) ; L.pointer_type (L.
     array_type i32_t 3) |] in
let drawPoint_func = L.declare_function "drawPoint" drawPoint_t
   the_module in
```

```
(* Declare the built-in drawPoint(), which draws a point in SDL
   *)
let print_t = L.function_type i32_t
  [| L.pointer_type (L.array_type i32_t 2) ; L.pointer_type i8_t
     ; L.pointer_type (L.array_type i32_t 3) |] in
let print_func = L.declare_function "print" print_t the_module in

(* Declare the built-in setFramerate() function *)
let setFramerate_t = L.function_type void_t [| i32_t |] in
let setFramerate_func = L.declare_function "setFramerate"
   setFramerate_t the_module in

(* Declare the built-in getFramerate() function *)
let getFramerate_t = L.function_type i32_t [| |] in
let getFramerate_func = L.declare_function "getFramerate"
   getFramerate_t the_module in

(* Declare the built-in allocTranslateArray(), which allocates
   space for displacements to shapes*)
let allocTranslateArrayX_t = L.function_type (L.pointer_type
   i32_t)
  [| L.pointer_type i32_t ; L.pointer_type i32_t; i32_t ; L.
     pointer_type i32_t|] in
let allocTranslateArrayX_func = L.declare_function "
   allocTranslateArrayX" allocTranslateArrayX_t the_module in

(* Declare the built-in allocTranslateArray(), which allocates
   space for displacements to shapes*)
let allocTranslateArrayY_t = L.function_type (L.pointer_type
   i32_t)
  [| L.pointer_type i32_t ; L.pointer_type i32_t; i32_t ; L.
     pointer_type i32_t|] in
let allocTranslateArrayY_func = L.declare_function "
   allocTranslateArrayY" allocTranslateArrayY_t the_module in

(* Declare the built-in translateCurve(), which translates a
   curve by some displacement in SDL *)
let translateCurve_t = L.function_type  void_t
  [|  L.pointer_type (L.array_type i32_t 2) ;  L.pointer_type (L.
     array_type i32_t 2) ;
      L.pointer_type (L.array_type i32_t 2) ; L.pointer_type
         i32_t ; L.pointer_type i32_t ; i32_t ; i32_t |] in
let translateCurve_func = L.declare_function "translateCurve"
   translateCurve_t the_module in

(* Declare the built-in translatePoint(), which translates a
   point by some displacement in SDL *)
```

```
let translatePoint_t = L.function_type  void_t
  [|  L.pointer_type (L.array_type i32_t 2) ; L.pointer_type
     i32_t ; L.pointer_type i32_t ; i32_t ; i32_t |] in
let translatePoint_func = L.declare_function "translatePoint"
   translatePoint_t the_module in

(* Define each function (arguments and return type) so we can
   call it *)
let function_decls =
  let function_decl m sfdecl =
    let name = sfdecl.S.sfname
    and formal_types =
      Array.of_list (List.map
        (fun (t,_) -> let ltyp = ltype_of_typ t in
          match t with
            A.Array(_) -> L.pointer_type (ltyp)
          | _ -> ltyp)
      sfdecl.S.sformals)
    in let ftype = (match name with
        "main" -> L.function_type i32_t formal_types
      | _ -> L.function_type (ltype_of_typ sfdecl.S.styp)
         formal_types) in
    StringMap.add name (L.define_function name ftype the_module,
       sfdecl) m in
  List.fold_left function_decl StringMap.empty functions in

(* Add in member functions for each shape *)
let function_decls =
  let shape_function_decl m ssdecl =
  let sname = ssdecl.S.ssname in
    let m = List.fold_left (fun m smember_f ->
      let f_name = smember_f.S.sfname
      and formal_types =
        Array.of_list (L.pointer_type (shape_type sname) ::
          List.map (fun (t,_) -> let ltyp = ltype_of_typ t in
            match t with
              A.Array(_) -> L.pointer_type (ltyp)
            | _ -> ltyp) smember_f.S.sformals)
      in let ftype = L.function_type (ltype_of_typ smember_f.S.
         styp) formal_types in
      StringMap.add f_name (L.define_function f_name ftype
         the_module, smember_f) m)
    m ssdecl.S.smember_fs in
    (* Add in each constructor and draw as well *)
    let construct_name = ssdecl.S.sconstruct.S.sfname and
    formal_types = Array.of_list (List.map (fun (t,_) -> let ltyp
       = ltype_of_typ t in
```

```
            match t with
              A.Array(_) -> L.pointer_type (ltyp)
            | _ -> ltyp)  ssdecl.S.sconstruct.S.sformals) in
      let ftype = L.function_type (L.pointer_type (shape_type sname
         )) formal_types in
      let m = StringMap.add construct_name (L.define_function
         construct_name ftype the_module, ssdecl.S.sconstruct) m in
      let draw_name = ssdecl.S.sdraw.S.sfname and
      formal_types = [| L.pointer_type (shape_type sname) |]
        in let ftype = L.function_type (void_t) formal_types in
      StringMap.add draw_name (L.define_function draw_name ftype
         the_module, ssdecl.S.sdraw) m in
    List.fold_left shape_function_decl function_decls shapes in

  (* Fill in the body of each shape type *)
  let shape_decl ssdecl =
    let name = ssdecl.S.ssname in
    let s_type = shape_type name in
    let lmember_vs = List.rev (List.fold_left (fun l (t, _) -> (
       ltype_of_typ t) :: l ) [] ssdecl.S.smember_vs) in
    let lmember_fs = List.rev (List.fold_left (fun l smember_f ->
      let formal_types =
        Array.of_list (List.map (fun (t,_) -> ltype_of_typ t)
           smember_f.S.sformals) in
      let ftype = L.function_type (ltype_of_typ smember_f.S.styp)
         formal_types in
       (L.pointer_type ftype) :: l ) [] ssdecl.S.smember_fs) in
    let lptr_members = [L.array_type i32_t max_translates; L.
       array_type i32_t max_translates;
      L.array_type i32_t max_translates; L.pointer_type i32_t; L.
         pointer_type i32_t; i32_t] in
    (L.struct_set_body s_type (Array.of_list(lmember_vs @
       lmember_fs @ lptr_members)) false) in

  ignore(List.iter shape_decl shapes);

  (* Fill in the body of the given function *)
  let build_function_body sfdecl member_vars =
    (* ignore(print_string (sfdecl.S.sfname ^ "\n")); *)
    let (the_function, _) = (try StringMap.find sfdecl.S.sfname
       function_decls
      with Not_found -> raise(Failure("build_function_body
         Not_found!"))) in
    let builder = L.builder_at_end context (L.entry_block
       the_function) in
```

```
    (* SPECIAL CASE: For the main(), add in a call to the
       initalization of the SDL window *)
    let _ = match sfdecl.S.sfname with
        "main" -> ignore(L.build_call startSDL_func [|  |] "
           startSDL" builder)
      | _ -> () in
    (* TODO: Consider storing the returned value somewhere, return
       that as an error *)

    let const_zero = L.const_int i32_t 0 in

    (* Construct the function's "locals": formal arguments and
       locally
       declared variables.  Allocate each on the stack, initialize
          their
       value, if appropriate, and remember their values in the "
          locals" map *)
    let local_vars =
      let add_formal m (t, n) p =
      (* Special case: for member variables, add the first formal (
         pointer to the instance) directly into the map *)
      if (String.length n > 2 && (String.sub n 0 2) = "__")
      then (StringMap.add n p m)
      else(
        L.set_value_name n p;
          let local =
            (match t with
            (* For arrays, use the pointer directly *)
              A.Array(_) -> p
            | _ -> let l = L.build_alloca (ltype_of_typ t) n
               builder in
                ignore (L.build_store p l builder); l) in
                StringMap.add n local m
      ) in

      let add_local m (t, n) =
        (* Special case: for constructors, don't re-add the local
           into the map *)
        if (String.length n > 2 && (String.sub n 0 2) = "__")
        then m
        else (
                let local_var = L.build_alloca (ltype_of_typ t) n
                   builder       in
          StringMap.add n local_var m
        ) in
```

```
    let formals = (List.fold_left2 add_formal StringMap.empty
       sfdecl.S.sformals
        (Array.to_list (L.params the_function)) )
    (* (* The only case where a mismatch occurs is for shape-
       member functions, when the first argument is the shape
    - in this case, ignore the first argument *)
    with Invalid_argument("List.fold_left2") -> List.fold_left2
       add_formal StringMap.empty sfdecl.S.sformals
        (List.tl (Array.to_list (L.params the_function))) *) in
    List.fold_left add_local formals sfdecl.S.slocals in

  (* Return the value for a variable or formal argument *)
  let lookup n = try StringMap.find n local_vars
                 with Not_found -> (try StringMap.find n
                    member_vars
                   with Not_found -> (try StringMap.find n
                      global_vars
                     with Not_found -> raise(Failure("lookup
                        Not_found!"))))
  in

  (* Construct code for an expression; return its value *)
  let rec expr builder loadval = function
          S.SInt_literal(i), _ -> L.const_int i32_t i
    | S.SFloat_literal(f), _ -> L.const_float f32_t f
    | S.SChar_literal(c), _ -> L.const_int i8_t (Char.code c)
    | S.SString_literal(s), _ -> L.build_global_stringptr s "tmp"
       builder
    | S.SNoexpr, _ -> const_zero
    | S.SArray_literal(_, s), (A.Array(_, prim_typ) as t) ->
        let const_array = L.const_array (ltype_of_typ prim_typ) (
           Array.of_list (List.map (fun e -> expr builder true e)
            s)) in
        if loadval then const_array
        else (let arr_ref = L.build_alloca (ltype_of_typ t) "
           arr_ptr" builder in
          ignore(L.build_store const_array arr_ref builder);
            arr_ref)
    | S.SArray_literal(_, _), _ -> raise(Failure("Invalid Array
       literal being created!"))
    | S.SBinop (e1, op, e2), _ ->
        let e1' = expr builder true e1
        and e2' = expr builder true e2 in
      (match op with
        S.IAnd -> L.build_and
          (conv_bool e1' builder) (conv_bool e2' builder) "tmp"
            builder
```

```
      | S.IOr -> L.build_or
          (conv_bool e1' builder) (conv_bool e2' builder) "tmp"
            builder
      | _ ->  (match op with
          S.IAdd     -> L.build_add
        | S.ISub     -> L.build_sub
        | S.IMult    -> L.build_mul
        | S.IDiv     -> L.build_sdiv
        | S.IMod     -> L.build_srem
        | S.IEqual   -> L.build_icmp L.Icmp.Eq
        | S.INeq     -> L.build_icmp L.Icmp.Ne
        | S.ILess    -> L.build_icmp L.Icmp.Slt
        | S.ILeq     -> L.build_icmp L.Icmp.Sle
        | S.IGreater-> L.build_icmp L.Icmp.Sgt
        | S.IGeq     -> L.build_icmp L.Icmp.Sge
        | S.FAdd     -> L.build_fadd
        | S.FSub     -> L.build_fsub
        | S.FMult    -> L.build_fmul
        | S.FDiv     -> L.build_fdiv
        | S.FMod     -> L.build_frem
        | S.FEqual   -> L.build_fcmp L.Fcmp.Oeq
        | S.FNeq     -> L.build_fcmp L.Fcmp.One
        | S.FLess    -> L.build_fcmp L.Fcmp.Olt
        | S.FLeq     -> L.build_fcmp L.Fcmp.Ole
        | S.FGreater-> L.build_fcmp L.Fcmp.Ogt
        | S.FGeq     -> L.build_fcmp L.Fcmp.Oge
        | _ -> raise(Failure("Found some binary operator that isn
            't handled!"))
        ) e1' e2' "tmp" builder
      )

    | S.SUnop(op, e), _ ->
          let e' = expr builder true e in
            (match op with
              S.INeg     -> L.build_neg e' "tmp" builder
      | S.INot     -> L.build_icmp L.Icmp.Eq (conv_bool (expr
         builder true e) builder) const_zero "tmp" builder
      | S.FNeg     -> L.build_fneg e' "tmp" builder)
    | S.SAssign (lval, s_e), _ -> let e' = expr builder true s_e
       in
                        ignore (L.build_store e' (lval_expr
                          builder lval) builder); e'
    | S.SCall (s_f, act), _ -> let f_name = s_f.S.sfname in
    let actuals = List.rev (List.map
      (fun (s_e, t) ->
        (* Send a pointer to array types instead of the actual
           array *)
```

```
          match t with
            A.Array(_) -> expr builder false (s_e, t)
          | _ -> expr builder true (s_e, t))
        (List.rev act)) in (* Why reverse twice? *)

      (match f_name with
          "consolePrint" -> let fmt_str_ptr =
              L.build_in_bounds_gep string_format_str [| const_zero
                  ; const_zero |] "tmp" builder in
            L.build_call printf_func (Array.of_list (fmt_str_ptr ::
                actuals)) "printf" builder
        | "intToString" -> let result = L.build_array_alloca i8_t (
           L.const_int i32_t 12) "intToString" builder in
            let int_fmt_ptr =
              L.build_in_bounds_gep int_format_str [| const_zero ;
                 const_zero |] "tmp" builder in
            ignore(L.build_call sprintf_func (Array.of_list (result
                :: int_fmt_ptr :: actuals)) "intToStringResult"
               builder);
            result
        | "floatToString" -> let result = L.build_array_alloca i8_t
            (L.const_int i32_t 20) "floatToString" builder in
            let flt_fmt_ptr =
              L.build_in_bounds_gep float_format_str [| const_zero
                 ; const_zero |] "tmp" builder in
            ignore(L.build_call sprintf_func (Array.of_list (result
                :: flt_fmt_ptr :: actuals)) "floatToStringResult"
               builder);
            result
        | "charToString" -> let result = L.build_array_alloca i8_t
           (L.const_int i32_t 2) "charToString" builder in
            let char_fmt_ptr =
              L.build_in_bounds_gep char_format_str [| const_zero ;
                  const_zero |] "tmp" builder in
            ignore(L.build_call sprintf_func (Array.of_list (result
                :: char_fmt_ptr :: actuals)) "charToStringResult"
               builder);
            result
        | "drawCurve" ->
            (* Add displacements to the actuals *)
            let disp_final_x = L.build_load (lookup "__disp_x") "
               load_disp_final_x" builder in
            let disp_final_y = L.build_load (lookup "__disp_y") "
               load_disp_final_y" builder in
            let num_frames = L.build_load (lookup "__num_frames") "
               load_num_frames" builder in
```

```
        let translateArgs = List.rev (List.tl (List.tl (List.
            rev actuals))) in
        ignore(L.build_call translateCurve_func
          (Array.of_list
            (List.rev(L.const_int i32_t (1) :: num_frames ::
               disp_final_y :: disp_final_x :: (List.rev
               translateArgs) )))
          "" builder);
        ignore(L.build_call drawCurve_func (Array.of_list(
            actuals)) "drawCurve_result" builder);
        (* Reverse the displacement *)
        L.build_call translateCurve_func
          (Array.of_list
            (List.rev(L.const_int i32_t (-1) :: num_frames ::
               disp_final_y :: disp_final_x :: (List.rev
               translateArgs) )))
          "" builder
      | "drawPoint" ->
      (* Add displacements to the actuals *)
        let disp_final_x = L.build_load (lookup "__disp_x") "
            load_disp_final_x" builder in
        let disp_final_y = L.build_load (lookup "__disp_y") "
            load_disp_final_y" builder in
        let num_frames = L.build_load (lookup "__num_frames") "
            load_num_frames" builder in
        let translateArgs = List.rev (List.tl (List.rev actuals
            )) in
        ignore(L.build_call translatePoint_func
          (Array.of_list
            (List.rev(L.const_int i32_t (1) :: num_frames ::
               disp_final_y :: disp_final_x :: translateArgs)))
          "" builder);
        ignore(L.build_call drawPoint_func (Array.of_list(
            actuals)) "drawPoint_result" builder);
        (* Reverse the displacement *)
        L.build_call translatePoint_func
          (Array.of_list
            (List.rev(L.const_int i32_t (-1) :: num_frames ::
               disp_final_y :: disp_final_x :: translateArgs)))
          "" builder
      | "print" ->
        let disp_final_x = L.build_load (lookup "__disp_x") "
            load_disp_final_x" builder in
        let disp_final_y = L.build_load (lookup "__disp_y") "
            load_disp_final_y" builder in
        let num_frames = L.build_load (lookup "__num_frames") "
            load_num_frames" builder in
```

```
            let translateArgs = List.rev (List.tl (List.tl (List.
               rev actuals))) in
            ignore(L.build_call translatePoint_func
              (Array.of_list
                (List.rev(L.const_int i32_t (1) :: num_frames ::
                   disp_final_y :: disp_final_x :: (List.rev(
                   translateArgs)))))
              "" builder);
            ignore(L.build_call print_func (Array.of_list(actuals))
                "print_result" builder);
            (* Reverse the displacement *)
            L.build_call translatePoint_func
              (Array.of_list
                (List.rev(L.const_int i32_t (-1) :: num_frames ::
                   disp_final_y :: disp_final_x :: (List.rev(
                   translateArgs)))))
              "" builder;
        | "setFramerate" -> L.build_call setFramerate_func (Array.
           of_list(actuals)) "" builder
        | "getFramerate" -> L.build_call getFramerate_func (Array.
           of_list(actuals)) "getFramerate_result" builder
        | "intToFloat" -> L.build_call intToFloat_func (Array.
           of_list(actuals)) "intToFloat_func" builder
        | "floatToInt" -> L.build_call floatToInt_func (Array.
           of_list(actuals)) "floatToInt_func" builder
        | "sine" -> L.build_call sine_func (Array.of_list(actuals))
            "sine_func" builder
        | "cosine" -> L.build_call cosine_func (Array.of_list(
           actuals)) "cosine_func" builder
        | "round" -> L.build_call round_func (Array.of_list(actuals
           )) "round_func" builder
        | "translate" ->
            (* Extract the first actual - this is the sequence
               number with which to add to the list of
               displacements *)
            let seq = (match List.hd act with
                (S.SInt_literal(_) , _ )as i -> expr builder true i
              | _ -> raise(Failure("Incorrect first argument for
                 translate!"))) in
            let act = List.tl act in
            let actuals = List.tl actuals in
            (* Extract the shape instance *)
            let (main_shape_inst, main_sname) = (match List.hd act
               with
                S.SLval(S.SId(n), _), A.Shape(sname) -> (lookup n,
                   sname)
```

```
            | _ -> raise(Failure("Incorrect first argument for
                translate!"))) in
          (* let act = List.tl act in *)
          let actuals = List.tl actuals in
          let child_shapes = find_child_objs [] main_shape_inst
             main_sname builder in
          ignore(List.iter (fun (inst_sname, shape_inst) ->
            let sdecl = shape_def inst_sname in
            let start_index = (List.length sdecl.S.smember_vs) +
               (List.length sdecl.S.smember_fs) in
            (* Get access to the correct elements to add in
               displacement and time values *)
            let disp_ref_x = L.build_gep
              (L.build_struct_gep shape_inst start_index "
                 disp_ref_x" builder)
              [| const_zero ; seq |] "disp_x_seq_ref" builder in
            let disp_ref_y = L.build_gep
              (L.build_struct_gep shape_inst (start_index + 1) "
                 disp_ref_y" builder)
              [| const_zero ; seq |] "disp_y_seq_ref" builder in
            let time_ref = L.build_gep
              (L.build_struct_gep shape_inst (start_index + 2) "
                 time_ref" builder)
              [| const_zero ; seq |] "disp_time_ref" builder in
                 (* const_zero *)
            (* Store values *)
            ignore(L.build_store (L.build_load
              (L.build_gep (List.hd actuals) [| const_zero ;
                 const_zero |] "disp_seq_ref" builder)
              "disp_val_x" builder ) disp_ref_x builder);
            ignore(L.build_store (L.build_load
              (L.build_gep (List.hd actuals) [| const_zero ; L.
                 const_int i32_t 1 |] "disp_seq_ref" builder)
              "disp_val_y" builder ) disp_ref_y builder);
            ignore(L.build_store (List.hd (List.rev actuals))
               time_ref builder);
          ) child_shapes); const_zero
      | _ -> let (fdef, fdecl), actuals =
          (try StringMap.find f_name function_decls, actuals
          with Not_found ->
          (* In this case, another member function is being
             called from inside a member function *)
          let (sname, inst_name) = (match List.hd sfdecl.S.
             sformals with
              (A.Shape(s), n) -> (s, n)
            | _ -> (* In this case, the member function is being
               called from the constructor *)
```

```
                match List.hd sfdecl.S.slocals with
                    (A.Shape(s), n) -> (s, n)
                  | _ -> raise(Failure("SCall Not_found"))
          )
          in
          let new_f_name = sname ^ "__" ^ f_name in
          (StringMap.find new_f_name function_decls, (lookup
             inst_name) :: actuals)) in
           let result = (match fdecl.S.styp with A.Void -> ""
                                               | _ -> f_name ^ "
                                                  _result") in
      L.build_call fdef (Array.of_list actuals) result builder)
  | S.SShape_fn(s, styp, s_f, act), _ -> let obj = (lookup s)
     in
      let f_name = (match styp with
          A.Shape(sname) -> sname
        | _ -> raise(Failure("Non-shape type object in member
           function call!"))) ^ "__" ^ s_f.S.sfname in
      let actuals = List.rev (List.map
        (fun (s_e, t) ->
          (* Send a pointer to array types instead of the
             actual array *)
          match t with
            A.Array(_) -> expr builder false (s_e, t)
          | _ -> expr builder true (s_e, t))
          (List.rev act)) in
      let (fdef, fdecl) = (try StringMap.find f_name
         function_decls with Not_found -> raise(Failure("
         SShape_fn Not_found!"))) in
      let result = (match fdecl.S.styp with A.Void -> ""
                                          | _ -> f_name ^ "
                                             _result") in
      L.build_call fdef (Array.of_list (obj :: actuals)) result
          builder
  | S.SLval(l), _ -> let lval = lval_expr builder l in
      if loadval then L.build_load lval "tmp" builder
      else lval
  | S.SInst_shape(_, sactuals), A.Shape(sname) ->
      let actuals =
      List.rev (List.map (fun (s_e, t) -> let ll_expr = expr
         builder true (s_e, t) in
        (* Send a pointer to array types instead of the actual
           array *)
        match t with
          A.Array(_) -> let copy = L.build_alloca (ltype_of_typ
             t) "arr_copy" builder in
            ignore(L.build_store ll_expr copy builder); copy
```

```
          | _ -> ll_expr)
          (List.rev sactuals)) in
        (* Call the constructor *)
        let (constr, _) = (try StringMap.find (sname ^ "
           __construct") function_decls with Not_found -> raise(
           Failure("SInst_shape Not_found!"))) in
        let new_inst = L.build_call constr (Array.of_list actuals
           ) (sname ^ "_inst_ptr") builder in
        L.build_load new_inst (sname ^ "_inst") builder
    | S.SInst_shape(_, _), _ -> raise(Failure("Cannot instantiate
       a shape of non-shape type!"))


  and lval_expr builder = function
    S.SId(s), _ -> lookup s
  | S.SAccess(id, idx), _(* el_typ *) ->
      (* ignore(print_string "access"); *)
      let arr = lookup id in
      let idx' = expr builder true idx in
      (* let arr_len = L.array_length (ltype_of_typ el_typ) in
      if (idx' < const_zero || idx' >= (L.const_int i32_t arr_len
         ))
        then raise(Failure("Attempted access out of array bounds
           "))
        (* TODO: figure out how to check for access out of array
           bounds *)
        else  *)L.build_gep arr [| const_zero ; idx' |] "tmp"
           builder
      (*let id' = lookup id
      and idx' = expr builder idx in
      if idx' < (expr builder (A.Int_literal 0)) || idx' > id
         '.(1) then raise(Failure("Attempted access out of array
         bounds"))
      else L.const_int i32_t idx'*)
  | S.SShape_var(s, v), s_t ->
      let rec resolve_shape_var obj var obj_type =
        (* Find index of variable in the shape definition *)
        match obj_type with
            A.Shape(sname) -> let sdef = shape_def sname in
            (match var with
                S.SId(v_n), _ -> let index = index_of (fun (_,
                   member_var) -> v_n = member_var) sdef.S.
                   smember_vs 0 in
                  L.build_struct_gep obj index "tmp" builder
              | S.SAccess(v_n, idx), _ -> let index = index_of (
                 fun (_, member_var) -> v_n = member_var) sdef.S.
                 smember_vs 0 in
```

```
                    let arr = L.build_struct_gep obj index "tmp"
                       builder in
                    let idx' = expr builder true idx in
                    L.build_gep arr [| const_zero ; idx' |] "tmp"
                       builder
                | S.SShape_var(member_n, member_v), member_t ->
                    let index = index_of (fun (_, member_var) ->
                       member_n = member_var) sdef.S.smember_vs 0
                       in
                    let id = L.build_struct_gep obj index "tmp"
                       builder in
                    resolve_shape_var id member_v member_t
              )
            | _ -> raise(Failure("Cannot access a shape variable of
                a non-shape type object!"))
        in
        resolve_shape_var (lookup s) v s_t

    in

    (* Invoke "f builder" if the current block doesn't already
       have a terminal (e.g., a branch). *)
    let add_terminal builder f =
      match L.block_terminator (L.insertion_block builder) with
              Some _ -> ()
      | None -> (* ignore(print_string "Found no return statement
          !");  *)ignore (f builder) in

    (* Build the code for the given statement; return the builder
       for
       the statement's successor *)
    let rec stmt builder = function
              S.SBlock sl -> List.fold_left stmt builder sl
      | S.SExpr e -> ignore (expr builder true e); builder
      (* | S.SVDecl ((t, n), e) -> let var = L.build_alloca (
         ltype_of_typ t) n builder in
          let e' = expr builder e in
          ignore(L.build_store e' var builder); builder *)
      | S.SReturn e -> ignore (match sfdecl.S.styp with
                A.Void -> L.build_ret_void builder
              | _ -> L.build_ret (expr builder true e) builder);
                 builder
      | S.SIf (predicate, then_stmt) ->
          let bool_val = conv_bool (expr builder true predicate)
             builder in
```

```
        let merge_bb = L.append_block context "merge"
           the_function in

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

        ignore (L.build_cond_br bool_val then_bb merge_bb builder
           );
        L.builder_at_end context merge_bb

    | S.SWhile (predicate, body) ->
        let pred_bb = L.append_block context "while" the_function
            in
        ignore (L.build_br pred_bb builder);

        let body_bb = L.append_block context "while_body"
           the_function in
        add_terminal (stmt (L.builder_at_end context body_bb)
           body)
          (L.build_br pred_bb);

        let pred_builder = L.builder_at_end context pred_bb in
        let bool_val = conv_bool (expr pred_builder true
           predicate) builder in

        let merge_bb = L.append_block context "merge"
           the_function in
        ignore (L.build_cond_br bool_val body_bb merge_bb
           pred_builder);
        L.builder_at_end context merge_bb
    | S.SShape_render(s, sname, num_t, sl) -> let main_shape_inst
       = lookup s in
        let child_shapes = find_child_objs [] main_shape_inst
           sname builder in
        let builder = List.fold_left stmt builder sl in
        (* For each child shape *)
        ignore(List.iter (fun (inst_sname, shape_inst) ->
          (* Call the external functions for allocTranslateArray
             to get references to the displacements to perform,
             per frame *)
          (* First get references to the correct data within the
             instance *)
          let sdecl = shape_def inst_sname in
```

```
            let start_index = (List.length sdecl.S.smember_vs) + (
               List.length sdecl.S.smember_fs) in
            (* ignore(print_string(inst_sname ^ " " ^ (
               string_of_int start_index) ^ "\n")); *)
            let disp_ref_x = L.build_gep
              (L.build_struct_gep shape_inst start_index "
                 disp_ref_x" builder)
              [| const_zero ; const_zero|] "fst_disp_ref_x" builder
                 in
            let disp_ref_y = L.build_gep
              (L.build_struct_gep shape_inst (start_index + 1) "
                 disp_ref_y" builder)
              [| const_zero ; const_zero|] "fst_disp_ref_y" builder
                 in
            let time_ref = L.build_gep
              (L.build_struct_gep shape_inst (start_index + 2) "
                 time_ref" builder)
              [| const_zero ; const_zero|] "fst_time_ref" builder
                 in
            let disp_final_x_ref = (L.build_struct_gep shape_inst (
               start_index + 3) "disp_final_x" builder) in
            let disp_final_y_ref = (L.build_struct_gep shape_inst (
               start_index + 4) "disp_final_y" builder) in
            let num_frames = (L.build_struct_gep shape_inst (
               start_index + 5) "num_frames_ref" builder) in
            ignore(L.build_store (L.build_call
               allocTranslateArrayX_func
              [| disp_ref_x ; time_ref ; L.const_int i32_t num_t ;
                 num_frames |]
              "allocX_result" builder) disp_final_x_ref builder);
            ignore(L.build_store (L.build_call
               allocTranslateArrayY_func
              [| disp_ref_y ; time_ref ; L.const_int i32_t num_t ;
                 num_frames |]
              "allocY_result" builder) disp_final_y_ref builder);
            (* let int_fmt_ptr =
                L.build_in_bounds_gep int_format_str [| const_zero
                   ; const_zero |] "tmp" builder in
            ignore(L.build_call printf_func [| int_fmt_ptr ; L.
               build_load (num_frames) "num_frames_loaded" builder
               |] "" builder); *)
          ) child_shapes);
          builder

    in

    (* Build the code for each statement in the function *)
```

```
    let builder = stmt builder (S.SBlock sfdecl.S.sbody) in

    (* SPECIAL CASE: For the main(), add in a call to the main
       rendering of the SDL window, return its result *)
    let _ = match sfdecl.S.sfname with
        "main" ->
        (* Find all shape objects within scope *)
        let final_objs = List.rev (List.fold_left (fun lst (t, n)
           -> match t with
            A.Shape(sname) -> let inst = lookup n in
              find_child_objs lst inst sname builder
          | A.Array(size, A.Shape(sname)) -> let arr_inst = lookup
             n in
              let ind_list = List.rev (List.fold_left
                (fun l i -> (L.build_gep arr_inst [| const_zero ; L
                   .const_int i32_t i |] "inst" builder) :: l)
                [] (range 0 (size - 1))) in
              List.fold_left (fun l ind_inst -> find_child_objs l
                 ind_inst sname builder) lst ind_list
          | _ -> lst)
        [] (List.rev sfdecl.S.slocals)) in

        (* Create a loop while program is running *)
        let pred_bb = L.append_block context "while" the_function
           in
        ignore (L.build_br pred_bb builder);

        let body_bb = L.append_block context "while_body"
           the_function in
        let builder_body = L.builder_at_end context body_bb in
        (* Build call to onRenderStartSDL() *)
        ignore(L.build_call onRenderStartSDL_func [|  |] ""
           builder_body);

        (* Build a call to each shape's draw function *)
        List.iter (fun (n, o) ->
          let (draw_fn, _) = StringMap.find (n ^ "__draw")
             function_decls in
          ignore(L.build_call draw_fn [| o |] "" builder_body) )
        final_objs;

        (* Build call to onRenderFinishSDL() *)
        ignore(L.build_call onRenderFinishSDL_func [|  |] ""
           builder_body);
        ignore(L.build_br pred_bb builder_body);

        let pred_builder = L.builder_at_end context pred_bb in
```

```
        let merge_bb = L.append_block context "merge" the_function
           in
        ignore (L.build_cond_br
          (L.build_load (StringMap.find "_Running" global_vars) "
             _Running_val" pred_builder)
        body_bb merge_bb pred_builder);
        let builder = L.builder_at_end context merge_bb in

        (* (* Free all allocated memory *)
        List.iter (fun (n, o) ->
          let sdecl = shape_def n in
          let start_index = (List.length sdecl.S.smember_vs) + (
             List.length sdecl.S.smember_fs) in
          let disp_x_ref = L.build_load (L.build_struct_gep o (
             start_index + 3) "disp_x" builder) "disp_x_load"
             builder in
          let disp_y_ref = L.build_load (L.build_struct_gep o (
             start_index + 4) "disp_y" builder) "disp_y_load"
             builder in
          ignore(L.build_free disp_x_ref builder);
          ignore(L.build_free disp_y_ref builder);
          ) final_objs; *)

        let stopSDL_ret = L.build_alloca i32_t "stopSDL_ret"
           builder in
          ignore(L.build_store (L.build_call stopSDL_func [|  |] "
             stopSDL_ret" builder) stopSDL_ret builder);
          ignore(L.build_ret (L.build_load stopSDL_ret "stopSDL_ret
             " builder) builder)
      | _ -> () in

    (* Add a return if the last block falls off the end *)
    (* add_terminal builder (match sfdecl.S.styp with
        A.Void -> L.build_ret_void
      | _ -> L.build_ret const_zero(* L.build_ret (L.const_int (
         ltype_of_typ t) 0) *)) *)
    match sfdecl.S.styp with
        A.Void -> add_terminal builder L.build_ret_void
      | _ -> ()
in

let build_object_function_body sfdecl sdecl =
  let sname = sdecl.S.ssname in
  let stype = shape_type sname in
  let (the_function, _) = (try StringMap.find sfdecl.S.sfname
     function_decls
```

```
    with Not_found -> raise(Failure("build_object_function_body
       Not_found!"))) in
  let builder = L.builder_at_end context (L.entry_block
     the_function) in
  let construct_name = sname ^ "__construct" in
  let inst_name = "__" ^ sname ^ "_inst" in

  let shape_inst =
    if sfdecl.S.sfname = construct_name
    (* SPECIAL CASE: For the construct(), add creation of an
       object of the required type *)
    then let inst = L.build_alloca stype inst_name builder in
      let maxFrameIndex = (List.length sdecl.S.smember_vs) + (
         List.length sdecl.S.smember_fs) + 5 in
      ignore(L.build_store (L.const_int i32_t 0) (L.
         build_struct_gep inst maxFrameIndex "num_frames" builder
         ) builder); inst
      (* In all other cases, return the first argument of the
         function *)
    else let inst = Array.get (L.params the_function) 0 in ignore
       (L.set_value_name inst_name inst); inst
  in

  let sfdecl =
    if sfdecl.S.sfname = construct_name
    then {sfdecl with S.styp = A.Shape(sname); S.slocals = (A.
       Shape(sname), inst_name) :: sfdecl.S.slocals}
    else {sfdecl with S.sformals = (A.Shape(sname), inst_name) ::
        sfdecl.S.sformals}
  in

  (* Create pointers to all member variables *)
  let member_vars = List.fold_left
    (fun m ((_, n), i) -> let member_val = L.build_struct_gep
       shape_inst i n builder in
      StringMap.add n member_val m)
    StringMap.empty (List.mapi (fun i v -> (v, i)) sdecl.S.
       smember_vs) in
  let start_index = (List.length sdecl.S.smember_vs) + (List.
     length sdecl.S.smember_fs) in
  (* Add allocations for pointer members *)
  let member_vars = StringMap.add "__disp_vals_x" (L.
     build_struct_gep shape_inst start_index "disp_vals_x"
     builder) member_vars in
  let member_vars = StringMap.add "__disp_vals_y" (L.
     build_struct_gep shape_inst (start_index + 1) "disp_vals_y"
     builder) member_vars in
```

```
    let member_vars = StringMap.add "__time_vals" (L.
        build_struct_gep shape_inst (start_index + 2) "time_vals"
        builder) member_vars in
    let member_vars = StringMap.add "__disp_x" (L.build_struct_gep
        shape_inst (start_index + 3) "disp_x" builder) member_vars
        in
    let member_vars = StringMap.add "__disp_y" (L.build_struct_gep
        shape_inst (start_index + 4) "disp_y" builder) member_vars
        in
    let member_vars = StringMap.add "__num_frames" (L.
        build_struct_gep shape_inst (start_index + 5) "num_frames"
        builder) member_vars in
    let member_vars = if sfdecl.S.sfname = construct_name
      then StringMap.add inst_name shape_inst member_vars
      else member_vars
    in

    (* Build rest of the function body *)
    build_function_body sfdecl member_vars;

    (* SPECIAL CASE: For the construct(), return the instantiated
        object *)
    if sfdecl.S.sfname = construct_name
    then let bbs = L.basic_blocks the_function in
      let builder = L.builder_at_end context (Array.get bbs ((Array
          .length bbs) - 1)) in
      (* build_function_body would have inserted a void return
          statement at the end; remove this *)
      match L.block_terminator (L.insertion_block builder) with
        Some ins -> (L.delete_instruction ins);
          ignore(L.build_ret shape_inst builder)
      | None -> ignore(L.build_ret shape_inst builder)
    else ()

  in

  List.iter (fun f -> build_function_body f StringMap.empty)
     functions;
  List.iter (fun s ->
    build_object_function_body s.S.sconstruct s;
    build_object_function_body s.S.sdraw s;
    List.iter (fun f -> build_object_function_body f s) s.S.
        smember_fs;)
  shapes;
  the_module
```

A.8 predefined.h

```
/*
 * @author: Kunal Baweja
 */
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <stdint.h>
#include <math.h>
#include <unistd.h>

#include "SDL2_gfxPrimitives.h"
#include "SDL2_imageFilter.h"
#include "SDL2_framerate.h"
#include "SDL2_rotozoom.h"

typedef struct {
    long int frame_interval; // In microseconds
    int curr_frame;
    SDL_Window* window;
    SDL_Renderer* renderer;
    SDL_Event Event;
} GAME;

/* Global variables for graphics management */
GAME theGame;
/* Boolean to keep track of whether the program is still running */
extern bool _Running;
FPSmanager fpsmanager;

int startSDL();
bool onInitSDL();
bool LoadContent();
void onEventSDL(SDL_Event* Event);
void clearSDL();
void onRenderStartSDL();
void onRenderFinishSDL();
int stopSDL();

/* type conversion statements */
double intToFloat(int num);
int floatToInt(double num);

/* trigonometric functions */
double sine(double angle);
double cosine(double angle);
```

```
/* Framerate functions */
int setFramerate(int rate);
int getFramerate();

/* Internal Draw functions of SOL */

bool drawPointUtil(const int point[2], const int rgb[3], const int
   opacity);
bool drawPoint(const int point[2], const int rgb[3]);

bool drawCurveUtil(const Sint16 *vx, const Sint16 *vy, const int
   num,
    const int steps, const int rgb[2], const int opacity);

bool drawCurve(const int start[2], const int mid[2], const int end
   [2],
    const int steps, const int rgb[3]);

/* print on SDL window; returns 0 on success, -1 on failure */
int print(const int pt[2], const char *text, const int color[3]);

/* rotate a coordinate */
void rotateCoordinate(int pt[2], const int axis[2], const double
   degree);

/* rotate a curve */
void rotateCurve(int start[2], int mid[2], int end[2], const int
   axis[2],
    const double degree);

/* translate a point */
void translatePoint(int pt[2], int* displaceX, int* displaceY, int
   maxFrame, int sign);

/* translateCurve */
void translateCurve(int start[2], int mid[2], int end[2],
    int* displaceX, int* displaceY, int maxFrame, int sign);

/* Allocate space based on multiple translates */
int* allocTranslateArrayX(int* indivDispX, int* times, int num, int
   * numFrames);
int* allocTranslateArrayY(int* indivDispY, int* times, int num, int
   * numFrames);
```

A.9 predefined.c

```
/*
 * @author: Kunal Baweja
 * Pre-defined functions for SOL
 */

#include "predefined.h"

bool onInitSDL() {
    if(SDL_Init(SDL_INIT_EVERYTHING) < 0) {
        return false;
    }

    if((theGame.window = SDL_CreateWindow("Shape Oriented Language"
       ,100,100,640, 480, SDL_WINDOW_SHOWN)) == NULL) {
        return false;
    }
    //SDL Renderer
    theGame.renderer = SDL_CreateRenderer(theGame.window, -1,
       SDL_RENDERER_ACCELERATED | SDL_RENDERER_PRESENTVSYNC);
    if (theGame.renderer == NULL){
        printf("%s \n", SDL_GetError());
        return 1;
    }
    theGame.curr_frame = 0;
    return true;
}

void onEventSDL(SDL_Event* Event) {
    if(Event->type == SDL_QUIT) {
        _Running = false;
    }
}

void clearSDL()
{
    /* clear screen before drawing again */
    SDL_SetRenderDrawColor(theGame.renderer, 242, 242, 242, 255);
    SDL_RenderClear(theGame.renderer);
}

void onRenderStartSDL() {
    while(SDL_PollEvent(&theGame.Event)) {
        onEventSDL(&theGame.Event);
    }
```

```
    clearSDL();
}

void onRenderFinishSDL()
{
    SDL_RenderPresent(theGame.renderer);
    // Enforce frame rate by sleeping
    usleep(theGame.frame_interval);
    theGame.curr_frame += 1;
}

int stopSDL()
{
    SDL_DestroyRenderer(theGame.renderer);
    SDL_DestroyWindow(theGame.window);
    SDL_Quit();
    return 0;
}

int startSDL() {

    theGame.window = NULL;
    _Running = true;

    if(onInitSDL() == false) {
        return -1;
    }

    /* initialize frame rate manager */
    SDL_initFramerate(&fpsmanager);
    // Set default values for framerate
    setFramerate(30);
    theGame.frame_interval = 33333;

    return 0;
}

/* draw a point in SOL */
bool drawPointUtil(const int point[2], const int rgb[3], const int
   opacity) {
    pixelRGBA(theGame.renderer, (Sint16)point[0], (Sint16)point[1],
        (Uint8)rgb[0], (Uint8)rgb[1], (Uint8)rgb[2], opacity);
    return true;
}

bool drawPoint(const int point[2], const int rgb[3]) {
    return drawPointUtil(point, rgb, 255);
```

```
}

/* helper function to draw a bezier curve in SOL */
bool drawCurveUtil(const Sint16 *vx, const Sint16 *vy, const int
   num,
    const int steps, const int rgb[3], const int opacity) {

    // pass arguments to SDL gfx
    bool res = bezierRGBA(theGame.renderer, vx, vy, num, steps, (
       Uint8)rgb[0],
        (Uint8)rgb[1], (Uint8)rgb[2], (Uint8)opacity);

    return res;
}

/* draw a bezier curve with 3 control points */
bool drawCurve(const int start[2], const int mid[2], const int end
   [2],
    int steps, const int rgb[3]) {
    // printf("(%d, %d), (%d, %d), (%d, %d), %d, (%d, %d, %d)\n",
    //     start[0], start[1], mid[0], mid[1], end[0], end[1],
       steps, rgb[0], rgb[1], rgb[2]);

    const int num = 3;

    Sint16 *vx = NULL;
    Sint16 *vy = NULL;

    // accumulate x and y coordinates
    if ((vx = (Sint16*)malloc(num * sizeof(Sint16))) == NULL)
        return false;

    if ((vy = (Sint16*)malloc(num * sizeof(Sint16))) == NULL) {
        free(vx);
        return false;
    }

    // x coordinates
    vx[0] = start[0];
    vx[1] = mid[0];
    vx[2] = end[0];

    // y coordinates
    vy[0] = start[1];
    vy[1] = mid[1];
    vy[2] = end[1];
```

```
    bool res = drawCurveUtil(vx, vy, num, steps, rgb, 255);

    // memory cleanup
    free(vx);
    free(vy);

    return res;
}

/* return double representation of int */
double intToFloat(int num){
    return (double)num;
}

/* return int representation of a double */
int floatToInt(double num) {
    return (int)num;
}

/* sine of angle accepted in degrees */
double sine(double angle) {
    return sin(angle * M_PI / 180.0);
}

/* cosine of angle accepted in degrees */
double cosine(double angle) {
    return cos(angle * M_PI / 180);
}

/*
 * set frames per second (positive integer)
 * returns 0 for sucess and -1 for error
 */
int setFramerate(int rate) {
    theGame.frame_interval = 1e6 / rate;
    return SDL_setFramerate(&fpsmanager, (Uint32)rate);
}


/* get current frame ratre per second */
int getFramerate() {
    return SDL_getFramerate(&fpsmanager);
}

/*
 * print on SDL window
 * returns 0 on success, -1 on failure
```

```
 */
int print(const int pt[2], const char *text, const int color[3]) {
    return stringRGBA(theGame.renderer, (Sint16)pt[0], (Sint16)pt
       [1], text,
        (Uint8)color[0], (Uint8)color[1], (Uint8)color[2], 255);
}

/*
 * rotate a coordinate clockwise by degree
 * around the axis point
 */
void rotateCoordinate(int pt[2], const int axis[2], const double
   degree) {
    // account for actual rotation to perform
    int _d = (int)(degree * 100);
    double _degree = _d / 100.0;
    _degree *= M_PI / 180.0;

    // translate back to origin
    pt[0] -= axis[0];
    pt[1] -= axis[1];

    // rotate and round off to nearest integers
    pt[0] = (int)nearbyint(pt[0] * cos(_degree) - pt[1] * sin(
       _degree));
    pt[1] = (int)nearbyint(pt[0] * sin(_degree) + pt[1] * cos(
       _degree));

    // translate point back
    pt[0] += axis[0];
    pt[1] += axis[1];
}

/* rotate a bezier curve control points */
void rotateCurve(int start[2], int mid[2], int end[2], const int
   axis[2],
    const double degree) {
    rotateCoordinate(start, axis, degree);
    rotateCoordinate(mid, axis, degree);
    rotateCoordinate(end, axis, degree);
}

/* translate a point by given displacement */
void translatePoint(int pt[2], int* displaceX, int* displaceY, int
   maxFrame, int sign) {
    if (maxFrame > 0) {
```

```
        pt[0] += (sign * displaceX[((theGame.curr_frame < maxFrame)
            ? theGame.curr_frame : maxFrame - 1)]);
        pt[1] += (sign * displaceY[((theGame.curr_frame < maxFrame)
            ? theGame.curr_frame : maxFrame - 1)]);
    }
}

/* translate a bezier curve control points */
void translateCurve(int start[2], int mid[2], int end[2],
    int* displaceX, int* displaceY, int maxFrame, int sign) {
    translatePoint(start, displaceX, displaceY, maxFrame, sign);
    translatePoint(mid, displaceX, displaceY, maxFrame, sign);
    translatePoint(end, displaceX, displaceY, maxFrame, sign);
}

int* allocTranslateArrayX(int* indivDispX, int* times, int num, int
   * numFrames) {
    int totalSeconds = 0;
    int i, j;
    int frameRate = getFramerate();
    int currTime, currFrames;
    for(i = 0; i < num; i++)
        totalSeconds += times[i];
    *numFrames = totalSeconds * frameRate;
    int* dispX = (int*) malloc((*numFrames) * sizeof(int));
    // printf("%d\n", dispX);

    float cumulX = 0.0;
    int frameIndex = 0;

    for(i = 0; i < num; i++) {
        currTime = times[i];
        currFrames = currTime * frameRate;
        // printf("%d, %d : %d\n", indivDispX[i], currTime, num);
        float dispPerFrameX = (float)indivDispX[i] / currFrames;
        for(j = 0; j < currFrames; j++) {
            cumulX += dispPerFrameX;
            dispX[frameIndex] = (int) cumulX;
            // printf("%d, ", (int)cumulX);
            frameIndex += 1;
        }
        // printf("\n");
    }
    // printf("%d, %d, %d, %d\n", frameIndex, (int) cumulX, dispX
       [0], dispX);
    return dispX;
}
```

```
int* allocTranslateArrayY(int* indivDispY, int* times, int num, int
   * numFrames) {
    int totalSeconds = 0;
    int i, j;
    int frameRate = getFramerate();
    int currTime, currFrames;
    for(i = 0; i < num; i++)
        totalSeconds += times[i];
    *numFrames = totalSeconds * frameRate;
    int* dispY = (int*) malloc((*numFrames) * sizeof(int));
    // printf("%d\n", dispY);

    float cumulY = 0.0;
    int frameIndex = 0;

    for(i = 0; i < num; i++) {
        currTime = times[i];
        currFrames = currTime * frameRate;
        // printf("%d, %d : %d\n", indivDispY[i], currTime, num);
        float dispPerFrameY = (float)indivDispY[i] / currFrames;
        for(j = 0; j < currFrames; j++) {
            cumulY += dispPerFrameY;
            dispY[frameIndex] = (int) cumulY;
            // printf("%d, ", (int)cumulY);
            frameIndex += 1;
        }
        // printf("\n");
    }
    // printf("%d, %d, %d, %d\n", frameIndex, (int) cumulY, dispY
       [0], dispY);

    return dispY;
}
```

A.10 Makefile

```
# @author: Kunal Baweja

# Make sure ocamlbuild can find opam-managed packages: first run
# eval `opam config env`
# Easiest way to build: using ocamlbuild, which in turn uses
   ocamlfind

CC = gcc
```

```
CFLAGS = -std=c99 -O2 -D_REENTRANT -I/usr/include/SDL2 -Wno-
   implicit
LIBS =
LFLAGS = -lSDL2 -lSDL2_gfx -lm

all : sol.native predefined.o

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

sol.d.byte:
        ocamlbuild -use-ocamlfind -pkgs llvm,llvm.analysis -cflags
           -w,+a-4 \
                sol.d.byte

# "make clean" removes all generated files

.PHONY : clean

clean:
        ocamlbuild -clean
        rm -rf testall.log *.diff sol scanner.ml parser.ml parser.
           mli
        rm -rf *.cmx *.cmi *.cmo *.cmx *.o *.s *.ll *.out *.exe *.
           err *.diff

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

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

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

scanner.ml : scanner.mll
        ocamllex scanner.mll

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

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

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

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

predefined.o: predefined.c
        $(CC) -c $^ $(CFLAGS) $(LIBS) $(LFLAGS)

# Testing the "bindings" example

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

Appendix B

Environment Setup

The following scripts can be used for installing dependencies and setting up environment.

B.1 install-llvm.sh

```
#!/bin/bash

#@author: Kunal Baweja

# update repo data
sudo apt update

# install llvm runtime and m4 preprocessor
sudo apt install --yes llvm-3.8-dev \
    llvm-3.8    \
    llvm-runtime\
    m4          \
    llvm

# install ocaml llvm binding
opam install --yes llvm.3.8
```

B.2 install-sdl-gfx.sh

```
#!/bin/bash

#@author: Kunal Baweja

SDL_GFX="SDL2_gfx-1.0.3"
SDL_GFX_TAR=$SDL_GFX".tar.gz"

# install sdl
sudo apt install --yes libegl1-mesa-dev \
    libgles2-mesa-dev\
```

```
    sdl2-2.0          \
    libsdl2-dev       \
    xdotool


# untar the file folder
tar xvzf $SDL_GFX_TAR

# step into directory
cd $SDL_GFX

# generate
./autogen.sh

# configure
./configure --prefix=/usr

# make
make

# install
sudo make install
```

Appendix C

Automated testing

The first two scripts are used for automated testing on Travis CI. For individual test cases, the author names are mentioned as first line of each test case.

C.1 .travis.yml

```
# @author: Kunal Baweja

language: c

sudo: required

os:
  - linux

env:
  - OCAML_VERSION=4.02

before_install:
  - wget https://raw.githubusercontent.com/ocaml/ocaml-ci-scripts/
     master/.travis-ocaml.sh
  - wget http://www.ferzkopp.net/Software/SDL2_gfx/SDL2_gfx-1.0.3.
     tar.gz

install:
  - bash -ex .travis-ocaml.sh
  - bash -ex install-llvm.sh
  - bash -ex install-sdl-gfx.sh

before_script:
  - eval `opam config env`
  - "export DISPLAY=:99.0"
  - "/sbin/start-stop-daemon --start --quiet --pidfile /tmp/
     custom_xvfb_99.pid --make-pidfile --background --exec /usr/bin
     /Xvfb -- :99 -ac -screen 0 1280x1024x24"
```

```
  - sleep 3

script:
  - make clean all
  - ./testall.sh
  - cat testall.log

notifications:
  email: false
```

C.2 testall.sh

```
#!/bin/bash

#@author: Kunal Baweja

# Regression testing script for sol
# 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 sol compiler.  Usually "./sol.native"
# Try "_build/sol.native" if ocamlbuild was unable to create a
   symbolic link.

SOL="./sol.native"

LIB="predefined.o"

SDL_FLAGS="-lSDL2 -lSDL2_gfx -lm"

# 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] [.sol 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"
}

# close sdl window
closeWindow() {
    # sleep 2 && xdotool key --clearmodifiers --delay 100 alt+F4
    xdotool sleep 2 && xdotool windowactivate --sync $(xdotool
       search --name "Shape Oriented Language") key --
       clearmodifiers --delay 100 alt+F4
}

# 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
    if [[ "$1" == *exe && "$1" != *mnl-* ]]; then
        closeWindow &
    fi
```

```
    eval $* || {
            SignalError "$1 failed on $*"
            return 1
    }
}

# RunFail <args>
# Report the command, run it, and expect an error
# Command may fail, we do not enforce by SignalError
# if it does not fail here
RunFail() {
    echo $* 1>&2
    if [[ "$1" == *exe ]]; then
        closeWindow &
    fi
    eval $* && {
        error=1
        return 1
    }
    return 0
}

Check() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
                              s/.sol//'`
    reffile=`echo $1 | sed 's/.sol$//'`
    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 ${basename}.out" &&
    Run "$SOL" "$1" ">" "${basename}.ll" &&
    Run "$LLC" "${basename}.ll" ">" "${basename}.s" &&
    Run "$CC" "-o" "${basename}.exe" "${basename}.s" "$LIB" "
       $SDL_FLAGS"&&
    Run "./${basename}.exe" ">" "${basename}.out" &&
    Compare ${basename}.out ${reffile}.gold ${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/.sol//'`
    reffile=`echo $1 | sed 's/.sol$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "$basename..."

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

    generatedfiles="${basename}.ll ${basename}.s ${basename}.err ${
       basename}.exe"
    RunFail "$SOL" "$1" "1>" "${basename}.ll" "2>" "${basename}.err
       "
    if [ $error -eq 1 ];
    then
        Run "$LLC" "${basename}.ll" "1>" "${basename}.s" &&
        Run "$CC" "-o" "${basename}.exe" "${basename}.s" "$LIB" "
           $SDL_FLAGS" &&
        RunFail "./${basename}.exe" "1>" "${basename}.err" "2>" "${
           basename}.err"
        error=0
    fi
    Compare ${basename}.err ${reffile}.err ${basename}.diff

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

```
        globalerror=$error
    fi
}

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

shift `expr $OPTIND - 1`

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

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

if [ ! -f predefined.o ]
then
    echo "Could not find predefined.o"
    echo "Try \"make clean all\""
    exit 1
fi

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

# ignore unknown files
for file in $files
do
    case $file in
        *test-*.sol|*mnl-*.sol)
            Check $file 2>> $globallog
            ;;
```

```
        *fail-*.sol)
            CheckFail $file 2>> $globallog
            ;;
    esac
done

exit $globalerror
```

C.3 fail-array-assign.sol

```
/*@author: Erik Dyer and Kunal Baweja*/

func main() {
    int [5]arr;
    int i;
    string s;

    /* array upper bound checking */
    i = 0;
    while(i < 6) {
        arr[i] = i;
        i = i + 1;
    }
}
```

C.4 test-array-of-shape.sol

```
/* @author: Kunal Baweja */

/* Test arrays of shapes */

/* Define a line */
shape Line {
    int [2]start;
    int [2]mid;
    int [2]end;

    construct(int [2]first, int [2]second) {
        start = first;
        end = second;
        /* line mid point */
        mid[0] = (start[0] + end[0]) / 2;
        mid[1] = (start[1] + end[1]) / 2;
    }
```

```
    draw(){}

    func describe() {
        consolePrint(intToString(start[0]));
        consolePrint(intToString(start[1]));
        consolePrint(intToString(mid[0]));
        consolePrint(intToString(mid[1]));
        consolePrint(intToString(end[0]));
        consolePrint(intToString(end[1]));
    }
}

func main() {

    /* four lines to describe a square */
    Line [4]sq;
    Line tmp;
    int i;

    /* describe four sides */
    sq[0] = shape Line([1,1], [3,1]); /* top */
    sq[1] = shape Line([3,1], [3,3]); /* right */
    sq[2] = shape Line([3,3], [1,3]); /* bottom */
    sq[3] = shape Line([1,3], [1,1]); /* left */

    /* print end and midpoit of each side */
    i = 0;
    while (i < 4) {
        tmp = sq[i];
        tmp.describe();
        i = i + 1;
    }
}
```

C.5 test-char-to-string.sol

```
/*@author: Kunal Baweja*/

func main() {
    char c;
    string s;

    c = 'h';
    s = charToString(c);
    consolePrint(s);
}
```

C.6 fail-div-semantic.sol

```
/*@author: Kunal Baweja*/

func main() {
    /* fail: numerator and denominator of different types */
    float x;
    x = 1.0 / 3;
}
```

C.7 test-add.sol

```
/*@author: Erik Dyer & Kunal Baweja*/

func int add(int x, int y) {
    return x + y;
}

func float fadd(float x, float y) {
    return x + y;
}

func main() {
    int x;
    float y;

    /* integer addition */
    x = add(40, 2);
    if (x == 42) {
        consolePrint("CORRECT");
    }
    if (x != 42) {
        consolePrint("INCORRECT");
    }


    /* float addition */
    y = fadd(38.0, 4.0);
    if (y == 42.0) {
        consolePrint("CORRECT");
    }
    if (y != 42.0) {
        consolePrint("INCORRECT");
```

```
    }
}
```

C.8 test-precedence.sol

```
/*@author: Kunal Baweja*/

func checkEqual(int x, int y) {
    if (x == y) {
        consolePrint("CORRECT");
    }
    if (x != y) {
        consolePrint("INCORRECT");
    }
}

func main() {
    int x;

    x = 1 + 20 * 3;     /* 61 */
    checkEqual(x, 61);

    x = 1 - 20 * 3;      /* -59 */
    checkEqual(x, -59);

    x = 1 + 18 / 3;      /* 7 */
    checkEqual(x, 7);

    x = 1 - 18 / 3;      /* -5 */
    checkEqual(x, -5);

    /* parenthesis override */
    x = (1 + 5) / 3;     /* 2 */
    checkEqual(x, 2);

    x = (1 - 7) / 3;     /* -2 */
    checkEqual(x, -2);

    /* for same precedence left to right associativity */
    x = 1 - 7 + 3;
    checkEqual(x, -3);

    x = 30 / 3 * 2;
    checkEqual(x, 20);

    /* unary negation precedes other arithmetic operators*/
```

```
    x = 3 + -2;
    checkEqual(x, 1);

    x = 3 - -2;
    checkEqual(x, 5);

    x = 3 * -2;
    checkEqual(x, -6);

    x = 3 / -1;
    checkEqual(x, -3);
}
```

C.9 test-if.sol

```
/*@author: Kunal Baweja*/

func main() {
    if (1) {
        consolePrint("INSIDE IF BLOCK");
    }
}
```

C.10 fail-prod-semantic.sol

```
/*@author: Kunal Baweja*/

func main() {
    int x;
    x = 1.0 * 1;    /* can not multiply float to int */
}
```

C.11 test-empty-function.sol

```
/*@author: Kunal Baweja*/

func empty(){}

func main(){
    consolePrint("BEFORE");
    empty();
    consolePrint("AFTER");
}
```

C.12 test-shape-member-shape.sol

```
/* @author: Kunal Baweja */

/* Test member shapes */

/* Define a line */
shape Line {
    int [2]start;
    int [2]end;

    construct(int [2]first, int [2]second) {
        start = first;
        end = second;
    }

    draw(){}

    func describe() {
        consolePrint(intToString(start[0]));
        consolePrint(intToString(start[1]));
        consolePrint(intToString(end[0]));
        consolePrint(intToString(end[1]));
    }

    func move(int [2]d) {
        int i;
        i = 0;
        while (i < 2) {
            start[i] = start[i] + d[i];
            end[i] = end[i] + d[i];
            i = i + 1;
        }
    }
}

/* Define rectange as a collection of lines */
shape Rectangle {
    Line top;
    Line right;
    Line bottom;
    Line left;

    construct(Line t, Line r, Line b, Line l) {
        top = t;
        right = r;
        bottom = b;
```

```
        left = l;
    }

    draw(){}

    func move(int [2]d) {
        top.move(d);
        right.move(d);
        bottom.move(d);
        left.move(d);
    }

    func describe() {
        top.describe();
        right.describe();
        bottom.describe();
        left.describe();
    }
}

func main() {
    Line t;
    Line b;
    Line r;
    Line l;
    Rectangle sq;

    /* define lines */
    t = shape Line([1,1], [3,1]);
    r = shape Line([3,1], [3,3]);
    b = shape Line([3,3], [1,3]);
    l = shape Line([1,3], [1,1]);

    /* initialize square */
    sq = shape Rectangle(t,r,b,l);
    sq.describe();

    /* move square */
    consolePrint("MOVE [2,2]");
    sq.move([2,2]);

    /*confirm all members are called*/
    sq.describe();
}
```

C.13 fail-array-access-pos.sol

```
/*@author: Erik Dyer*/

func main() {
    int i;
    int [5]array;

    /* assign array elements */
    array = [0,1,2,3,4];

    /* print array elements */
    i = 0;
    consolePrint(intToString(array[5]));
}
```

C.14 fail-parameter-floatint.sol

```
/*@author: Kunal Baweja*/

func add(int x, int y) {
    return x + y;
}

func main() {
    int x;
    x = add(40, 2.5); /* Fail: passing a float to a func that
       expects int */
}
```

C.15 fail-recursion.sol

```
/*@author: Kunal Baweja*/

/* find sum of 1 to n, inclusive */
func int series(int n) {
    /* fail no terminating condition */
    return n + series(n-1);
}

func main() {
    /* crash due to stack overflow */
    consolePrint(intToString(series(-1)));  /*-1*/
}
```

C.16 test-while.sol

```
/*@author: Kunal Baweja*/

func main() {
    int x;
    x = 5;
    while (x > 0) {
        consolePrint("INSIDE WHILE");
        x = x - 1;
    }
}
```

C.17 fail-return-void-int.sol

```
/*@author: Erik Dyer*/

func somefun() {
    return 42; /* Fail: return int from void function */
}

func main() {
   somefun();
}
```

C.18 test-function-shape-formal.sol

```
/* @author: Kunal Baweja */

/* Shapes should be passed by reference */

/* Define a line */
shape Line {
    int [2]start;
    int [2]end;

    construct(int [2]first, int [2]second) {
        start = first;
        end = second;
    }

    draw(){}

    func describe() {
        consolePrint(intToString(start[0]));
```

```
        consolePrint(intToString(start[1]));
        consolePrint(intToString(end[0]));
        consolePrint(intToString(end[1]));
    }
}

/* displace line by d */
func moveLine(Line l, int[2] d) {
    l.start[0] = l.start[0] + d[0];
    l.start[1] = l.start[1] + d[1];

    l.end[0] = l.end[0] + d[0];
    l.end[1] = l.end[1] + d[1];
}

func main() {
    Line l;
    l = shape Line([1,1], [5,5]);
    moveLine(l, [2, 2]);

    /* confirm modified values for pass by reference */
    l.describe();
}
```

C.19 test-product.sol

```
/*@author: Kunal Baweja*/

func int mult(int x, int y) {
    return x * y;
}

func float fmult(float x, float y) {
    return x * y;
}

func checkInt(int x, int y) {
    if (x == y) {
        consolePrint("CORRECT");
    }
    if (x != y) {
        consolePrint("INCORRECT");
    }
}

func checkFloat(float x, float y) {
```

```
    if (x == y) {
        consolePrint("CORRECT");
    }
    if (x != y) {
        consolePrint("INCORRECT");
    }
}

func main() {
    int x;
    float y;

    /* integer multiplication */
    x = mult(40, 2);
    checkInt(x, 80);

    x = mult(1, 0);
    checkInt(x, 0);

    /* float multiplication */
    y = fmult(-3.0, 2.0);
    checkFloat(y, -6.0);

    y = fmult(0.0, 1.0);
    checkFloat(y, 0.0);
}
```

C.20 fail-array-access-neg.sol

```
/*@author: Erik Dyer*/

func main() {
    int i;
    int [5]array;

    /* assign array elements */
    array = [0,1,2,3,4];

    /* print array elements */
    consolePrint(intToString(array[-1]));
}
```

C.21 test-logical.sol

```
/*@author: Kunal Baweja*/

func main() {

    int x;
    x = 1;

    if (x == 1 && x == 1) {
        consolePrint("AND");
    }
    if (1 == 1 || 1 == 0) {
        consolePrint("OR");
    }
    if (!(1 == 0)) {
        consolePrint("NOT");
    }
}
```

C.22 test-escape-chars.sol

```
/* @author: Kunal Baweja */

/* test escape sequences */

func main() {
    consolePrint("\\");
    consolePrint("new\nline");
    consolePrint("\'");
    consolePrint("\"");
    consolePrint("\ttableft");
}
```

C.23 fail-return-int-string.sol

```
/*@author: Erik Dyer*/

func int somefun() {
    return "should return int";
}

func main() {
    somefun();
}
```

C.24 test-array-pass-ref.sol

```
/*@author: Kunal Baweja*/
/* test arrays passed by reference */

func assign(int [5]b) {
    int i;
    i = 0;

    while(i < 5) {
        i = b[i] = i + 1;
    }
}

func main() {
    int [5]a;
    int i;

    /* pass for assignment */
    assign(a);

    /* confirm assigned values */
    i = 0;
    while (i < 5) {
        consolePrint(intToString(a[i]));
        i = i + 1;
    }
}
```

C.25 test-int-to-string.sol

```
/*@author: Kunal Baweja*/

func main() {
    string s;

    s = intToString(-2147483648);
    consolePrint(s);

    s = intToString(-2147483648 + 2147483647);
    consolePrint(s);

    s = intToString(0);
    consolePrint(s);

    s = intToString(2147483647);
```

```
    consolePrint(s);
}
```

C.26 test-shape-function.sol

```
/* @author: Kunal Baweja*/
/* confirm return values of member functions */

shape Rectangle {
    int [2]a;
    int [2]b;
    int [2]c;
    int [2]d;

    construct (int [2]w, int[2]x, int[2]y, int[2]z) {
        a = w;
        b = x;
        c = y;
        d = z;
    }

    draw () {}

    /* get area */
    func int area() {
        int h;
        int w;

        h = a[1] - d[1];
        if (h < 0) {
            h = -h;
        }

        w = b[0] - a[0];
        if (w < 0) {
            w = -w;
        }

        return w * h;
    }
}

func main () {
    Rectangle r;
    r = shape Rectangle([1,1],[4,1],[4,4],[1,4]);
    consolePrint(intToString(r.area()));
```

```
}
```

C.27 test-float-to-string.sol

```
/*@author: Kunal Baweja*/

func main() {
    float f;
    string s;

    f = -10.0;
    s = floatToString(f);
    consolePrint(s);

    f = 0.0;
    s = floatToString(f);
    consolePrint(s);

    f = 10.0;
    s = floatToString(f);
    consolePrint(s);
}
```

C.28 test-framerate.sol

```
/* @author Kunal Baweja */

func main() {
    /* test set and get framerate functions */
    consolePrint(intToString(getFramerate()));
    setFramerate(24);
    consolePrint(intToString(getFramerate()));
}
```

C.29 fail-if.sol

```
/*@author: Kunal Baweja*/

func main() {
    /* if condition expects integer expression */
    if (1.0) {
        consolePrint("INVALID CONDITION");
    }
}
```

C.30 test-shape-array.sol

```
/* @author: Kunal Baweja */

/* test initializing a shape with an array of points
 * pass an array to the constructor and ensure that
 * the object makes a copy of the array. The contents
 * of array should not change
 */

shape Line {
    int [2]start;
    int [2]mid;
    int [2]end;

    construct(int [2]first, int [2]second) {
        start = first;
        end = second;
        /* line mid point */
        mid[0] = (start[0] + end[0]) / 2;
        mid[1] = (start[1] + end[1]) / 2;
    }

    draw(){}

    func describe() {
        consolePrint(intToString(start[0]));
        consolePrint(intToString(start[1]));
        consolePrint(intToString(mid[0]));
        consolePrint(intToString(mid[1]));
        consolePrint(intToString(end[0]));
        consolePrint(intToString(end[1]));
    }
}

func main() {
    Line l;
    int [2]first;
    int [2]second;

    first = [1, 1];
    second = [9, 9];

    l = shape Line(first, second);

    /* modify source array */
    first[0] = -1;
```

```
    first[1] = -1;
    second[0] = -9;
    second[1] = -9;

    /* verify shape remains unchanged */
    l.describe();
}
```

C.31 test-trigono-func.sol

```
/* @author: Kunal Baweja */

/* test trignonometric functions */

func main(){

    /* sine function */
    consolePrint(floatToString(sine(90.0)));
    consolePrint(floatToString(sine(180.0)));
    consolePrint(floatToString(sine(270.0)));
    consolePrint(floatToString(sine(-90.0)));

    /* cosine function */
    consolePrint(floatToString(cosine(90.0)));
    consolePrint(floatToString(cosine(180.0)));
    consolePrint(floatToString(cosine(270.0)));
    consolePrint(floatToString(cosine(0.0)));
}
```

C.32 fail-func-arg.sol

```
/* @author: Kunal Baweja */

/* type mismatch for function argument */

func main() {
    consolePrint(1);
}
```

C.33 fail-add-semantic.sol

```
/*@author: Kunal Baweja*/

func float add(int x, float y) {
```

```
    return x + y;
}

func main() {
    float x;
    x = add(40, 2.5);
}
```

C.34 test-hello.sol

```
/* @author: Erik Dyer */

func main() {
    consolePrint("Hello World");
}
```

C.35 fail-assign-stringint.sol

```
/*@author: Erik Dyer*/

func int add(int x, int y) {
    return x + y;
}

func main() {
    int x;
    string y;
    int z;
    y = "foo";
    x = add(10, 2);
    z = "bar"; /* cant assign string to int*/
}
```

C.36 test-assign-variable.sol

```
/*@author: Kunal Baweja*/

func main() {
    int x;
    int y;
    float f;
    float g;
    string s;
    string p;
```

```
    string q;

    /* integer assignment */
    x = 5;
    y = x;
    s = intToString(y);
    consolePrint(s);

    /* string variable assignment */
    p = "Hello World";
    q = p;
    consolePrint(q);

    f = 4.2;
    g = f;
    consolePrint(floatToString(g));
}
```

C.37 test-set-array.sol

```
/*@author: Erik Dyer*/

func main() {
    int[2] x;
    int[2] y;

    y[0] = 4;
    y[1] = 2;

    x = y;
    consolePrint(intToString(x[0]));
    consolePrint(intToString(x[1]));
}
```

C.38 test-array-assign.sol

```
/*@author: Kunal Baweja*/

func main() {
    int [5]arr;
    int i;
    string s;

    i = 0;
    while(i < 5) {
```

```
        arr[i] = i;
        i = i + 1;
    }

    i = 4;
    while(i >= 0) {
        s = intToString(arr[i]);
        consolePrint(s);
        i = i - 1;
    }
}
```

C.39 test-comparison.sol

```
/*@author: Kunal Baweja*/

func main() {
    /* Integer comparisons */
    if (0 == 0) {
        consolePrint("EQUALITY");
    }
    if (-1 != 0) {
        consolePrint("INEQUALITY");
    }
    if (2 > 1) {
        consolePrint("GREATER THAN");
    }
    if (-2 < -1) {
        consolePrint("LESS THAN");
    }
    if (1 <= 2) {
        consolePrint("LESS THAN OR EQUAL");
    }
    if (5 >= 3) {
        consolePrint("GREATER THAN OR EQUAL");
    }

    /* float logical comparison */
    if (0.0 == 0.0) {
        consolePrint("FLOAT EQUALITY");
    }
    if (-1.0 != 0.0) {
        consolePrint("FLOAT INEQUALITY");
    }
    if (2.0 > 1.0) {
        consolePrint("FLOAT GREATER THAN");
```

```
    }
    if (-1.1 < -1.0) {
        consolePrint("FLOAT LESS THAN");
    }
    if (1.0 <= 2.0) {
        consolePrint("FLOAT LESS THAN OR EQUAL");
    }
    if (5.0 >= 3.0) {
        consolePrint("FLOAT GREATER THAN OR EQUAL");
    }
}
```

C.40 fail-add-intstring.sol

```
/*@author: Erik Dyer*/

func int add(int x, int y) {
    return x + y;
}

func main() {
    float x;
    string y;
    y = "foo";
    x = add(40, y); /* cant add string and int */
}
```

C.41 test-recursion.sol

```
/*@author: Kunal Baweja*/

/* find sum of 1 to n, inclusive */
func int series(int n) {
    if (n < 2)
        return n;
    return n + series(n-1);
}

func main() {
    consolePrint(intToString(series(-1)));   /*-1*/
    consolePrint(intToString(series(0)));    /*0*/
    consolePrint(intToString(series(1)));    /*1*/
    consolePrint(intToString(series(2)));    /*3*/
    consolePrint(intToString(series(10)));   /*55*/
}
```

C.42 test-division.sol

```
/*@author: Kunal Baweja*/

func int div(int x, int y) {
    return x / y;
}

func float fdiv(float x, float y) {
    return x / y;
}

func checkInt(int x, int y) {
    if (x == y) {
        consolePrint("CORRECT");
    }
    if (x != y) {
        consolePrint("INCORRECT");
    }
}

func checkFloat(float x, float y) {
    if (x == y) {
        consolePrint("CORRECT");
    }
    if (x != y) {
        consolePrint("INCORRECT");
    }
}

func main() {
    int x;
    float y;

    /* integer diviplication */
    x = div(40, 2);
    checkInt(x, 20);

    x = div(2, 5);
    checkInt(x, 0);

    /* float division */
    y = fdiv(-4.0, 2.0);
    checkFloat(y, -2.0);

    y = fdiv(0.0, 1.0);
    checkFloat(y, 0.0);
```

```
}
```

C.43 test-associativity.sol

```
/*@author: Kunal Baweja*/

func main() {
    int x;
    x = 1 + 2 - 3;  /* 0 */
    if (x == 0) {
        consolePrint("CORRECT");
    }

    x = 21 * 3 % 80 / 9; /* 7 */
    if (x == 7) {
        consolePrint("CORRECT");
    }
}
```

C.44 test-math-round.sol

```
/* @author: Kunal Baweja */

/* Test math round function */

func main() {
    consolePrint(floatToString(round(1.9)));
    consolePrint(floatToString(round(-1.9)));
    consolePrint(floatToString(round(9.0)));
}
```

C.45 test-shape-define.sol

```
/*@author: Kunal Baweja*/

shape Circle{
    int [2]center;
    int radius;
    construct(int [2]c, int r) {
        center[0] = c[0];
        center[1] = c[1];
        radius = r;
    }
```

```
    draw() {}

    func describe() {
        consolePrint("Center X");
        consolePrint(intToString(center[0]));
        consolePrint("Center Y");
        consolePrint(intToString(center[1]));
        consolePrint("Radius");
        consolePrint(intToString(radius));
    }
}
func main() {
    Circle c;
    int a;

    c = shape Circle([3, 5], 5);
    c.describe();

    /* change member variables */
    c.center[0] = -3;
    c.center[1] = -5;
    c.radius = 30;
    c.describe();
}
```

C.46 test-array-access.sol

```
/*@author: Kunal Baweja*/

func main() {
    int i;
    int [5]array;

    /* assign array elements */
    array = [0,1,2,3,4];

    /* print array elements */
    i = 0;
    while(i < 5) {
        consolePrint(intToString(array[i]));
        i = i  + 1;
    }
}
```

C.47 test-float-to-int.sol

```
/* @author: Kunal Baweja */

/* float to int conversions */

func main() {
    int x;

    x = floatToInt(0.5);
    consolePrint(intToString(x));

    x = floatToInt(-0.5);
    consolePrint(intToString(x));

    x = floatToInt(0.0);
    consolePrint(intToString(x));

    x = floatToInt(1.5);
    consolePrint(intToString(x));

    x = floatToInt(-1.5);
    consolePrint(intToString(x));
}
```

C.48 test-int-to-float.sol

```
/* @author Kunal Baweja */

/* int to float type conversion */
func main() {
    float x;

    x = intToFloat(-1);
    consolePrint(floatToString(x));

    x = intToFloat(0);
    consolePrint(floatToString(x));

    x = intToFloat(2);
    consolePrint(floatToString(x));
}
```

Appendix D

Manual Testing

D.1 mnl-composite-square.sol

```
/* @author: Kunal Baweja */

/* Test member shapes */

/* Define a line */
shape Line {
    int [2]start;
    int [2]mid;
    int [2]end;
    int [3]color;

    construct(int [2]first, int [2]second, int [3]clr) {
        start = first;
        end = second;
        color = clr;

        mid[0] = (start[0] + end[0]) / 2;
        mid[1] = (start[1] + end[1]) / 2;
    }

    draw(){
        drawCurve(start, mid, end, 2, color);
    }

}

/* Define rectange as a collection of lines */
shape Rectangle {
    Line top;
    Line right;
    Line bottom;
    Line left;
```

```
    construct(int [2]a, int [2]b, int [2]c, int [2]d) {
        top = shape Line(a, b, [150, 0, 0]);
        right = shape Line(b, c, [0, 150, 0]);
        bottom = shape Line(c, d, [0, 0, 150]);
        left = shape Line(d, a, [150, 150, 150]);
    }

    draw(){}

}

func main() {
    Rectangle sq;

    /* define lines */

    /* initialize square */
    sq = shape Rectangle([10,10], [300,10], [300, 300], [10, 300]);
}
```

D.2 mnl-drawpoint.sol

```
/* @author: Kunal Baweja */

shape Points {
    int [2]a;

    construct (int [2]x) {
        a = x;
    }

    draw() {
        int i;
        int [2]b;

        b = a;
        i = 0;
        while (i < 50) {
            b[0] = b[0] + i;
            b[1] = b[1] + i;
            drawPoint(b, [0, 0, 0]);
            i = i + 1;
        }
    }
}
```

```
func main() {
    Points p;
    p = shape Points([100, 100]);
    setFramerate(2);
}
```

D.3 mnl-polygon.sol

```
/*@author: Kunal Baweja & Erik Dyer*/

/* Test drawing an n sided polygon */

shape Spokes{
    int [2]center;
    float sides;
    float radius;
    construct(int [2]s, int l, int n) {
        center = s;
        sides = intToFloat(n);
        radius = intToFloat(l);
    }

    draw(){
        float i;
        int x;
        int y;
        float degrees;
        int[2] mid;
        int[2] end;

        i = 1.0;
        while(i <= sides){
            degrees = (360.0 * i)/sides;
            x = floatToInt(radius * cosine(degrees));
            y = floatToInt(radius * sine(degrees));

            end[0] = x + center[0];
            end[1] = y + center[1];
            mid[0] = (center[0] + end[0]) / 2;
            mid[1] = (center[1] + end[1]) / 2;
            drawCurve(center, mid, end, 2, [150, 0, 0]);
            i = i + 1.0;
        }
    }
}
```

```
shape Polygon{
    int [2]center;
    float sides;
    float radius;
    Spokes spks;

    construct(int [2]s, int l, int n) {
        center = s;
        sides = intToFloat(n);
        radius = intToFloat(l);
        spks = shape Spokes(s, l, n);
    }

    draw(){
        float i;
        float degrees;
        int[2] strt;
        int[2] mid;
        int[2] end;

        degrees = 0.0;
        strt[0] = center[0] + floatToInt(radius * cosine(degrees));
        strt[1] = center[1] + floatToInt(radius * sine(degrees));

        i = 1.0;
        while(i <= sides){
            degrees = (360.0 * i)/sides;
            end[0] = center[0] + floatToInt(radius * cosine(degrees
               ));
            end[1] = center[1] + floatToInt(radius * sine(degrees))
               ;
            mid[0] = (strt[0] + end[0]) / 2;
            mid[1] = (strt[1] + end[1]) / 2;
            drawCurve(strt, mid, end, 2, [0, 150, 0]);
            strt = end;
            i = i + 1.0;
        }
    }
}


func main() {
    Polygon p;
    p = shape Polygon([320, 240], 120, 24);
}
```

D.4 mnl-disco-bar.sol

```
/* @author: Kunal Baweja */

/* Describe a dancing line */

shape DanceLine {
    int [2]start;
    int [2]end;
    int [3]color;
    int freq;   /* oscillation frequency */
    int cnt;
    int d;      /* length change per iteration */

    /* constructor */
    construct(int [2]s, int [2]e, int[3]clr, int f) {
        start = s;
        end = e;
        color = clr;
        freq = f;
        cnt = 0;    /* initial counter */
        d = 2;
    }

    /* drawing specification for single frame */
    draw(){
        int i;
        int [2]s;   /* control points for drawCurve */
        int [2]m;
        int [2]e;

        /* increment count on each frame */
        cnt = cnt + 1;

        if (cnt > freq) {
            d = -d;     /* reverse direction */
            cnt = 0;    /* reset freq counter */
        }

        /* change object length on one end */
        end[1] = end[1] + d;

        s = start;  /* end points of line */
        e = end;

        /* draw 10 bezier curves for thickness */
        i = 0;
```

```
        while (i < 10) {
            s[0] = s[0] + 1;
            e[0] = e[0] + 1;

            /* bezier curve mid point */
            m[0] = (s[0] + e[0]) / 2;
            m[1] = (s[1] + e[1]) / 2;

            /* draw straight bezier curve */
            drawCurve(s, m, e, 2, color);
            i = i + 1;
        }
    }
}

shape DiscoBars {
    DanceLine d1;   /* member variables */
    DanceLine d2;
    DanceLine d3;

    construct () {
        /* instantiate DanceLine member variables */
        d1 = shape DanceLine([100, 300], [100, 252], [30, 144,
           255], 20);
        d2 = shape DanceLine([130, 300], [130, 202], [210, 105,
           30], 40);
        d3 = shape DanceLine([160, 300], [160, 132], [50, 205, 50],
            80);
    }

    draw(){
        /* empty draw function
         * draw functions of member variables
         * are called implicitly
         */
    }
}

func main() {
    DiscoBars bars;
    bars = shape DiscoBars();
}
```

D.5 mnl-hello.sol

```
/* @author: Kunal Baweja */
```

```
shape Text {
    int [2]loc;
    int [3]clr;
    string text;

    construct(int [2]l, string t, int [3]c) {
        loc = l;
        clr = c;
        text = t;
    }

    draw(){
        print(loc, text, clr);
    }
}

func main() {
    Text t;
    t = shape Text([280, 240], "Welcome to SOL !", [0, 100, 100]);
}
```

D.6 mnl-line.sol

```
/*@author: Kunal Baweja*/

/* Test drawing a line */

shape Line {
    int [2]start;
    int [2]mid;
    int [2]end;

    construct(int [2]s, int [2]e) {
        start = s;
        end = e;
        mid[0] = (start[0] + end[0]) / 2;
        mid[1] = (start[1] + end[1]) / 2;
    }

    draw(){
        drawCurve(start, mid, end, 100, [0,150,0]);
    }
}
```

```
func main() {
    Line l;
    l = shape Line([2,2], [200,200]);
}
```

D.7 mnl-thick-line.sol

```
/*@author: Kunal Baweja*/

/* Test drawing a line */

shape Line {
    int [2]start;
    int [2]end;

    construct(int [2]s, int [2]e) {
        start = s;
        end = e;
    }

    draw(){
        int i;
        int [2]s;
        int [2]m;
        int [2]e;
        i = 0;

        s = start;
        e = end;
        while (i < 20) {
            s[0] = s[0] - 1;
            s[1] = s[1] + 1;

            e[0] = e[0] - 1;
            e[1] = e[1] + 1;

            m[0] = (s[0] + e[0]) / 2;
            m[1] = (s[1] + e[1]) / 2;

            drawCurve(s, m, e, 2, [0,150,0]);
            i = i + 1;
        }
    }
}
```

```
func main() {
    Line l;
    l = shape Line([100,2], [200,200]);
    setFramerate(10);
}
```

D.8 mnl-ferris-move.sol

```
/* @author: Kunal Baweja */

/* Test drawing an n sided polygon */

shape Spokes{
    int [2]center;
    int [3]color;
    float sides;
    float radius;
    float theta;

    construct(int [2]s, int l, int n, float t, int[3] clr) {
        center = s;
        color = clr;
        theta = t;
        sides = intToFloat(n);
        radius = intToFloat(l);
    }

    draw(){
        float i;
        int x;
        int y;
        float degrees;
        int[2] mid;
        int[2] end;

        i = 0.0;
        while(i < sides){
            degrees = (360.0 * i)/sides + theta;
            x = floatToInt(radius * cosine(degrees));
            y = floatToInt(radius * sine(degrees));

            end[0] = x + center[0];
            end[1] = y + center[1];
            mid[0] = (center[0] + end[0]) / 2;
            mid[1] = (center[1] + end[1]) / 2;
            drawCurve(center, mid, end, 2, color);
```

```
            i = i + 1.0;
        }
    }
}

shape Polygon{
    int [2]center;
    float sides;
    float radius;
    float theta;
    int [3]color;

    construct(int [2]s, int l, int n, float t, int[3] clr) {
        center = s;
        color = clr;
        theta = t;
        sides = intToFloat(n);
        radius = intToFloat(l);
    }

    draw(){
        float i;
        float degrees;
        int[2] strt;
        int[2] mid;
        int[2] end;

        degrees = theta;
        strt[0] = center[0] + floatToInt(radius * cosine(degrees));
        strt[1] = center[1] + floatToInt(radius * sine(degrees));

        i = 1.0;
        while(i <= sides){
            degrees = (360.0 * i)/sides + theta;
            end[0] = center[0] + floatToInt(radius * cosine(degrees
               ));
            end[1] = center[1] + floatToInt(radius * sine(degrees))
               ;
            mid[0] = (strt[0] + end[0]) / 2;
            mid[1] = (strt[1] + end[1]) / 2;
            drawCurve(strt, mid, end, 2, color);
            strt = end;
            i = i + 1.0;
        }
    }
}
```

```
shape Square {
    Polygon plgn;

    construct(int[2] ctr, int r) {
        plgn = shape Polygon(ctr, r, 4, 45.0, [0, 0, 150]);
    }

    draw(){}
}

shape FerrisWheel {
    int frames;
    float radius;
    int sides;
    int [2]center;

    Polygon plgn;
    Spokes spks;
    Square [10]s;

    construct(int[2] ctr, int r, int n) {
        int i;
        float degrees;
        int[2] strt;

        /* carriages rotate around a point below
         * the center of ferris wheel spokes
         */
        center[1] = ctr[1] + 16;
        center[0] = ctr[0];

        frames = 0;
        radius = intToFloat(r);
        sides = n;
        plgn = shape Polygon(ctr, r, n, 0.0, [160, 82, 45]);
        spks = shape Spokes(ctr, r, n, 0.0, [34, 139, 34]);

        i = 0;
        while(i < sides) {
            degrees = (360.0 * intToFloat(i)) / intToFloat(sides);
            strt[0] = ctr[0] + floatToInt(radius * cosine(degrees))
                + 2;
            strt[1] = ctr[1] + floatToInt(radius * sine(degrees)) +
                15;
            s[i] = shape Square(strt, 20);
            i = i + 1;
```

```
        }
    }

    draw(){
        int i;
        float xn;
        float yn;
        Square tmp;
        float deg;

        deg = 0.0;

        /* speed up */
        if (frames <= 100) {
            deg = intToFloat(frames / 10);
            frames = frames + 2;
        }

        /* constant speed */
        if (frames > 100 && frames <= 200) {
            frames = frames + 1;
            deg = 10.0;
        }

        /* decrease speed */
        if (frames > 200 && frames <= 300) {
            frames = frames + 2;
            deg = intToFloat((300 - frames) / 10);
        }

        plgn.theta = (plgn.theta + deg) % 360.0;
        spks.theta = (spks.theta + deg) % 360.0;

        i = 0;
        while (i < sides) {
            tmp = s[i];
            /* translate back to origin */
            tmp.plgn.center[0] = tmp.plgn.center[0] - center[0];
            tmp.plgn.center[1] = tmp.plgn.center[1] - center[1];

            /* rotate point*/
            xn = intToFloat(tmp.plgn.center[0]) * cosine(deg);
            xn = xn - intToFloat(tmp.plgn.center[1]) * sine(deg);
            xn = round(xn + intToFloat(center[0]));

            yn = intToFloat(tmp.plgn.center[0]) * sine(deg);
            yn = yn + intToFloat(tmp.plgn.center[1]) * cosine(deg);
```

```
            yn = round(yn + intToFloat(center[1]));

            tmp.plgn.center[0] = floatToInt(xn);
            tmp.plgn.center[1] = floatToInt(yn);
            s[i] = tmp;
            i = i + 1;
        }

        /* cheeky message */
        print([240, 420], "JAVANGERS Amusement Park !", [209, 125,
           99]);
    }
}

func main() {
    FerrisWheel p;
    p = shape FerrisWheel([320, 240], 120, 10);
    setFramerate(15);
}
```

D.9 mnl-triangle.sol

```
/*@author: Erik Dyer & Kunal Baweja */

/* Test Triangle*/
/* Run this code, */


func findCenter(int [2]m, int[2]x, int[2]y) {
    m[0] = (x[0] + y[0]) / 2;
    m[1] = (x[1] + y[1]) / 2;
}

shape Triangle {
    int[2] a;
    int[2] b;
    int[2] c;

    int[2] abm;
    int[2] bcm;
    int[2] acm;

    construct (int[2] a_init, int[2] b_init, int[2] c_init){
        int i;
        a = a_init;
        b = b_init;
```

```
        c = c_init;

        i = 0;
        while (i < 2) {
            abm[i] = (a[i] + b[i]) / 2;
            bcm[i] = (c[i] + b[i]) / 2;
            acm[i] = (a[i] + c[i]) / 2;
            i = i + 1;
        }

        findCenter(abm, a, b);
        findCenter(acm, a, c);
        findCenter(bcm, c, b);
    }

    draw() {
        /* Draw lines between the three vertices of the triangle*/
        drawCurve(a, abm, b, 2, [150, 0, 0]);
        drawCurve(b, bcm, c, 2, [0, 150, 0]);
        drawCurve(c, acm, a, 2, [0, 0, 150]);
    }
}

func main(){
        Triangle t;
        t = shape Triangle([170, 340], [470, 340], [320, 140]);

}
```

D.10 mnl-james-bond.sol

```
/*@author: Kunal Baweja & Erik Dyer*/

/* Test Triangle translate */

shape Polygon{
    int [2]center;
    float sides;
    float radius;
    float theta;
    int [3]color;

    construct(int [2]s, int l, int n, float t, int[3] clr) {
        center = s;
        color = clr;
        theta = t;
```

```
        sides = intToFloat(n);
        radius = intToFloat(l);
    }

    draw(){
        float i;
        float degrees;
        int[2] strt;
        int[2] mid;
        int[2] end;

        degrees = theta;
        strt[0] = center[0] + floatToInt(radius * cosine(degrees));
        strt[1] = center[1] + floatToInt(radius * sine(degrees));

        i = 1.0;
        while(i <= sides){
            degrees = (360.0 * i)/sides + theta;
            end[0] = center[0] + floatToInt(radius * cosine(degrees
               ));
            end[1] = center[1] + floatToInt(radius * sine(degrees))
               ;
            mid[0] = (strt[0] + end[0]) / 2;
            mid[1] = (strt[1] + end[1]) / 2;
            drawCurve(strt, mid, end, 2, color);
            strt = end;
            i = i + 1.0;
        }
    }
}

shape Line {
    int [2]start;
    int [2]mid;
    int [2]end;

    construct(int [2]s, int [2]e) {
        start = s;
        end = e;
        mid[0] = (start[0] + end[0]) / 2;
        mid[1] = (start[1] + end[1]) / 2;
    }

    draw(){
        drawCurve(start, mid, end, 2, [0, 0, 0]);
    }
}
```

```
shape Person {
    Polygon head;
    Line body;
    Line lleg;
    Line rleg;
    Line lhand;
    Line rhand;
    int frames;

    construct(int[2] pos) {
        int [2]strt;
        int [2]end;

        frames = 0;
        head = shape Polygon(pos, 12, 50, 0.0, [0, 0, 0]);

        strt[0] = pos[0];
        strt[1] = pos[1] + 12;
        end[0] = pos[0];
        end[1] = pos[1] + 42;
        body = shape Line(strt, end);

        strt[1] = end[1];
        end[0] = strt[0] - 10;
        end[1] = strt[1] + 10;
        lleg = shape Line(strt, end);

        end[0] = strt[0] + 10;
        rleg = shape Line(strt, end);

        /* rhand */
        strt[1] = pos[1] + 22;
        end[1] = pos[1] + 22;
        rhand = shape Line(strt, end);

        /* rhand */
        end[0] = end[0] - 20;
        lhand = shape Line(strt, end);
    }

    draw(){
        if (frames < 120) {
            frames = frames + 1;
        }
        if (frames >= 120) {
            print([-200, 200], "The name is Bond. James Bond !",
```

```
                [102, 102, 255]);
        }
    }
}

func main(){
    Polygon c1;
    Polygon c2;
    Polygon c3;
    Person bond;

    c1 = shape Polygon([-50, 100], 40, 80, 30.0, [0,0,150]);
    c2 = shape Polygon([-50, 100], 40, 80, 30.0, [0,150,0]);
    c3 = shape Polygon([-50, 100], 40, 80, 30.0, [150,0,0]);
    bond = shape Person([-50, 80]);


    bond.render = {
        translate([400, 0], 4);
    }

    c1.render = {
        translate([400, 0], 4);
    }

    c2.render = {
        translate([200, 0], 2);
    }

    c3.render = {
        translate([100, 0], 1);
    }
}
```

Appendix E

Commit Logs

```
commit 45a9e341bdd1b46eac2cb675674490fa8e0fb5df
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Tue Dec 19 15:11:16 2017 -0500

    project timeline

commit 04f0d01394ccc37b89eae90ac2035af5c0dbf4a2
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Tue Dec 19 13:42:40 2017 -0500

    update tutorial in report

commit be7e411812a7dc99df717ff7673c6ed46b55c18c
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Tue Dec 19 03:52:40 2017 -0500

    thick line test

commit 8fb59fe552a3c417eeced3b087578759737812d6
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Tue Dec 19 03:07:38 2017 -0500

    composite shape drawing

commit f5c2729c1a7f13f6ec159715b446f2434875c597
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Tue Dec 19 02:42:49 2017 -0500

    add manual tests

commit 6088b7619a89238557188d75f10a3a090b1a8d6c
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Tue Dec 19 02:15:12 2017 -0500
```

```
    Enabled drawing of composite shapes i.e. member variables of
       instantiated shapes will also be drawn

commit ca80b13f46acdc1b7086944a38b5898a171042ae
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Tue Dec 19 00:44:19 2017 -0500

    Enabled setFramerate/getFramerate; enforced this by sleeping
       after each render loop in SDL

commit c8a58382bbcdab11904334d4f0365ce75dac5265
Merge: 11b73eb f707b0f
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Mon Dec 18 23:49:44 2017 -0500

    Merge branch 'master' of https://github.com/bawejakunal/sol

commit 11b73ebf76e294e8a48242f4917a3cd57160d6af
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Mon Dec 18 23:49:26 2017 -0500

    Fixed issue with calling member functions from inside other
       member functions of the same shape; Fixed potential issue
       with constructors having no return statement

commit f707b0fd4e12dccce2f011f4c68573d210f08c71
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Mon Dec 18 22:47:43 2017 -0500

    triangle and line test

commit e2e50c3400ac55d239c8dbf2bc0a7ca1b0026c72
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Mon Dec 18 22:11:28 2017 -0500

    Implemented drawPoint and print

commit d0a7f572375c63af8d4698823585bdbb3beaaf01
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Mon Dec 18 20:47:41 2017 -0500

    Finished implementation of drawCurve

commit 3f9b55213f104a68c855b1dfc313f36fd9e07894
Merge: 33bf564 998c0fe
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Mon Dec 18 18:16:43 2017 -0500
```

```
    Merge branch 'master' of https://github.com/bawejakunal/sol
       into remove_array_ref

commit 33bf5647accbab519449567a22c60bfb0b1edece
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Mon Dec 18 18:15:51 2017 -0500

    Partial commit of drawing curves

commit 998c0fe8afb77f83ef05c94bb8160a186bb9eefd
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Mon Dec 18 15:38:22 2017 -0500

    recusion tests

commit 56d1b3146c7b3e6a9bf791cf4adba259870b1c8d
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sun Dec 17 20:50:25 2017 -0500

    removed deferencing in shape member functions

commit 34932c560f8ba68e974f066dfbd78bfa200bcfbc
Merge: 6a00705 0141e9b
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sun Dec 17 14:57:46 2017 -0500

    Merge branch 'master' of github.com:bawejakunal/sol

commit 6a00705fa73f471ce759d05318615be66007776c
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sun Dec 17 14:57:08 2017 -0500

    update lrm

commit 0141e9b93d4a8ba667a58beb0533cbe170259c2d
Author: DyerSituation <ead2174@columbia.edu>
Date:   Sun Dec 17 14:09:34 2017 -0500

    added test-set-array

commit ee7d243a3835614f7fb6f3d3b85799c0e3787084
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sun Dec 17 12:13:49 2017 -0500

    add lrm to report
```

```
commit b021782e7af8ae94d975691121dc8a409ffcd549
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sun Dec 17 00:17:51 2017 -0500

    update drawcurve signature

commit 6e85751fa1af5ff2282020247faee22703d99fce
Merge: 171b639 985c738
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Sun Dec 17 00:02:07 2017 -0500

    Merge branch 'master' of https://github.com/bawejakunal/sol
       into remove_array_ref

commit 171b6393c0582371693b10ddeb9361d5664d2af2
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Sun Dec 17 00:01:38 2017 -0500

    Fixed array references - now all arrays are constants, and only
        references are created when passing into functions

commit 985c73806b4405c8d546cc2928b6ef4787de1107
Merge: 2caa22c d2b10be
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Dec 16 22:21:02 2017 -0500

    resolve conflict

commit d2b10bebdd68721f3a0d3196d90f1682017c5aba
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Dec 16 20:12:12 2017 -0500

    code listings; report draft

commit 2caa22ca9c469d4b1e737bf222c0996eb71b664f
Merge: e6c58ce af53236
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Sat Dec 16 17:30:41 2017 -0500

    Merge branch 'master' of https://github.com/bawejakunal/sol

commit e6c58ce5e0f2234ffca8d5f4d1adef7b426c3c8a
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Sat Dec 16 17:30:31 2017 -0500

    Added ability to access array elements of a shape's member
       variable; Updated code to allocate space for member arrays
```

```
commit af53236c4339e4b46d7651def62de7c0a772077a
Author: Erik Dyer <ead2174@columbia.edu>
Date:   Sat Dec 16 17:02:18 2017 -0500

    remove render function from within triangle

commit 5617f536f1f569a552d2af18f0f639ce02963429
Author: Erik Dyer <ead2174@columbia.edu>
Date:   Sat Dec 16 17:01:41 2017 -0500

    Update test-triangle.sol

commit a1569e851a7f2614b82ddcc3c0337cedbdde8873
Author: Erik Dyer <ead2174@columbia.edu>
Date:   Sat Dec 16 16:57:52 2017 -0500

    Update test-triangle-translate.sol

commit 118f976c04a3106e4c81a6ffbd57b11f45b2aa25
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Dec 16 16:53:16 2017 -0500

    TODO: update shape examples in LRM

commit 7ac691e0e69de27b21851d4d8c58ad4c3749800c
Merge: d039159 464463b
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Dec 16 14:14:30 2017 -0500

    Merge branch 'testing'

commit d0391593658628ebef423d9258b19f43a7472a04
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Sat Dec 16 14:11:59 2017 -0500

    Converted all arrays to references; enabled pass-by-reference
       for arrays

commit 464463b61884aa428888d149aad4405e99cc8f31
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Dec 16 13:35:00 2017 -0500

    update *toString in LRM

commit 83a91b2270a49ffaa7eb424641aab63d78f0cd0d
Author: Kunal Baweja <bawejakunal15@gmail.com>
```

```
Date:   Sat Dec 16 13:09:18 2017 -0500

    repo clean and update readme

commit 6a3aa835b13b9abf9a84ab85ca5bf9dd1b19936c
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Dec 16 13:08:45 2017 -0500

    repo clean and update readme

commit baa886894c5b14671a419229cf523c5f198ba476
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Dec 16 12:10:01 2017 -0500

    shape array member access fails

commit 2951355c5cfbf3d2aacd7a749f24b37d82539107
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Dec 16 10:56:31 2017 -0500

    change window name

commit d5cb5ca95ea1974c565471371ecb29107b8b0591
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Fri Dec 15 23:50:26 2017 -0500

    update *toString in tests

commit b057a8e2cc30dd65ec0ab4b32062aaf9f4b898c0
Merge: 3e96dd4 827f404
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Fri Dec 15 23:21:36 2017 -0500

    Merge branch 'testing'

commit 3e96dd4f6c81c21318eb85b0d45dd8cfba24baeb
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Fri Dec 15 23:16:28 2017 -0500

    Reverted *ToString functions to the original format (single
       argument, return value string)

commit 3aed393a7742be578cd3a40a1a7f771db59c1fd7
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Fri Dec 15 23:03:23 2017 -0500

    Removed compile-time array bounds checking
```

```
commit 827f404a3f08c77b8391fbe3ceddc69a45174113
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Fri Dec 15 22:55:34 2017 -0500

    try array access errors

commit 7e99af59be6a659db5427c172abeac28c1bbba01
Merge: e949027 1c2042d
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Fri Dec 15 22:27:21 2017 -0500

    Merge branch 'master' into testing

commit e9490274d538f737e64f3dc301adb9c2ec48f385
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Fri Dec 15 22:25:52 2017 -0500

    divide by zero undefined

commit 1c2042dfd8142b1b65841119d8f33303277261c0
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Fri Dec 15 21:31:13 2017 -0500

    Corrected code for array bounds checking

commit fc00591501f2e5901e91fdc2aded4aef29149f03
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Fri Dec 15 21:24:07 2017 -0500

    Moved const format strings to global, to prevent multiple
       copies

commit 3ad5f139bfa3143b8f874677f121cfe7888e5eed
Merge: deedd74 7add364
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Fri Dec 15 20:58:26 2017 -0500

    Merge branch 'master' into testing

commit 7add364960760a22aeb844f40a9def3676bf6e20
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Fri Dec 15 20:39:12 2017 -0500

    Corrected code for using float comparisons as boolean
       expressions
```

```
commit 542c34ecf0f7fb8bc45081e3e9c484db1ab33bf8
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Fri Dec 15 20:38:33 2017 -0500

    Added code for enabling arithmetic and comparison operators for
        floats

commit 0289752e06ddb79197cd257976a270015e8bf22b
Merge: 40836c8 2cf0c74
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Fri Dec 15 16:33:26 2017 -0500

    Merge branch 'master' of https://github.com/bawejakunal/sol

commit 40836c835e40a3df48d27cf622cf4d8803fe85eb
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Fri Dec 15 16:33:11 2017 -0500

    Completed basic implementation of Shapes, Instantiation, Dot
       access for member variables and functions - drawing
       functions yet to be started

commit deedd749155161aeac4335685e4bc853dfae31bf
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Thu Dec 14 23:45:40 2017 -0500

    fixed files cleanup

commit 9c56d157cef1d483737e758fecdc6c518f04d5dc
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Thu Dec 14 22:08:22 2017 -0500

    todo: file cleanup

commit 110ada2e7f73ed6f475c637a326c7de79d046371
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Thu Dec 14 21:16:49 2017 -0500

    try xvfb start-stop daemon

commit e4857ea8e920d19af2918bcacf41332cf3eecc86
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Thu Dec 14 21:06:01 2017 -0500

    add xvfb screen

commit b310a9faf08982dd5c4499d838151bde5a20292c
```

```
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Thu Dec 14 20:59:22 2017 -0500

    update failure test runs

commit b3278dc3d398665d649cfbcad2710753d951c670
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Thu Dec 14 19:53:49 2017 -0500

    see test logs in travis

commit 787e1d7c093542248ad989f502f4e053a7b08f9a
Merge: 29cbaab 2cf0c74
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Thu Dec 14 19:53:14 2017 -0500

    Merge branch 'master' into testing

commit 2cf0c742b03dd0b9445cffd4461ac27bc7b6b217
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Thu Dec 14 19:39:27 2017 -0500

    use std=c99 for gcc

commit 938f284c202b6b4600fdc784258661b7d165eed1
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Thu Dec 14 19:29:37 2017 -0500

    try xdotool window close

commit 29cbaab1498fa0255fbe72d62c9237b67e05d773
Author: DyerSituation <ead2174@columbia.edu>
Date:   Thu Dec 14 18:36:53 2017 -0500

    added fail assign arrays

commit 62f87d4f9eb497d2e87ebbc9357da7ff58a3401b
Author: DyerSituation <ead2174@columbia.edu>
Date:   Thu Dec 14 18:32:26 2017 -0500

    added out of bounds access test

commit df059ff3e1134245ab2c80fcc7cd650ea5aad7b3
Merge: 0a243ac 1edb2f2
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Thu Dec 14 18:29:12 2017 -0500
```

```
    Merge branch 'master' of github.com:bawejakunal/sol

commit 0a243ac7c672f32bb425578bd325dced2dacddec
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Thu Dec 14 18:29:07 2017 -0500

    test empty function

commit 1edb2f2fe8afd38902c45a1d537f4785be702fa8
Author: DyerSituation <ead2174@columbia.edu>
Date:   Thu Dec 14 18:19:25 2017 -0500

    runSol now complete

commit e5e95c62fa61c5963556b2b646171978a3e3ebc1
Author: DyerSituation <ead2174@columbia.edu>
Date:   Thu Dec 14 17:34:44 2017 -0500

    tweaked runSol.sh

commit 4ba3feb4bc369bdccc5cd65b207e314f31a5a232
Author: Erik Dyer <ead2174@columbia.edu>
Date:   Thu Dec 14 17:10:40 2017 -0500

    Create runSOL.sh

    script used to streamline running sol files

commit 449a8b1b6cb1b184d28a43941b0d9aa058a3c038
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Thu Dec 14 16:56:30 2017 -0500

    update order of precedence rules

commit 0f66cbfa8278f177e90f60fb488be7edeb6c4c1f
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Wed Dec 13 21:37:04 2017 -0500

    fixed size arrays

commit f60c5d2f373602c13a7beea70af9e347ef4ba565
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Wed Dec 13 20:51:14 2017 -0500

    add type conversion functions to LRM

commit 8d1dfe8ac417d1d1b10589b4f47a577e4a4849aa
```

```
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Wed Dec 13 20:16:51 2017 -0500

    fix typos

commit e2ba7def6d39752a319485ed58cffb0521abfe4f
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Wed Dec 13 20:10:31 2017 -0500

    add unary operator; fix order of precedence

commit 08a0c172a8653280768f587d8becf8460fbc2384
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Wed Dec 13 18:09:40 2017 -0500

    add array assignment test

commit cbbcc237078862d12c14610b93f70cc3f4c73074
Merge: 0ba8bb2 75ee170
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Wed Dec 13 18:02:45 2017 -0500

    Merge branch 'testing'

commit 0ba8bb2278663d76fd999e8cea180b7f766d5880
Merge: b5784a8 381bc99
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Wed Dec 13 17:58:31 2017 -0500

    Merge branch 'master' of https://github.com/bawejakunal/sol

commit b5784a8360aaace150c0e42f3f6bb583777bf0e5
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Wed Dec 13 17:58:13 2017 -0500

    Modified structure of AST and SAST to have a new type of
       expression - lvalue, which allows values to be assigned to
       it; added Id and Array Access into this lvalue; commented
       out array bounds checking in codegen as it was not behaving
       as expected

commit 75ee170fa1c4ee28e306f481385f196609b2d32e
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Wed Dec 13 17:07:51 2017 -0500

    test variable assignment
```

```
commit 381bc9987d05e9485710503192f67e0475c43a9d
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Wed Dec 13 16:37:22 2017 -0500

    add type conversion tests

commit 1f460b3beecd879e89b233c147baa402f1b3baca
Merge: b90821f 1e74a58
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Wed Dec 13 15:44:02 2017 -0500

    Merge branch 'master' into testing

commit 1e74a58953666eb97ccd9fdbb754f6d444a1d485
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Wed Dec 13 15:41:57 2017 -0500

    Added implementations of toString functions; changed
       implementation of float in codegen into double

commit b90821f1f3b2f7d571a1d35096c022cfb4fa9735
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Fri Dec 8 22:08:59 2017 -0500

    drawCurve steps argument

commit 79424d906553155d3e091b033368ec71bda7beac
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Thu Dec 7 17:48:05 2017 -0500

    print screen function

commit 9d393731c01c60532a06f99229f73bd7764412f7
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Wed Dec 6 19:44:44 2017 -0500

    try graphics travis

commit 95eb7abd8dab729be2d081665ba31e33155acb24
Merge: 5f6c3b7 675cc7b
Author: DyerSituation <ead2174@columbia.edu>
Date:   Wed Dec 6 19:41:10 2017 -0500

    Merge branch 'master' of https://github.com/bawejakunal/sol

commit 5f6c3b7c051675ca51ef4e2989656aac1c2c985e
Author: DyerSituation <ead2174@columbia.edu>
```

```
Date:   Wed Dec 6 19:40:43 2017 -0500

    added a translate test for triangle

commit 8d9551e5983ee385343e6071093fdb06a02b25b6
Author: DyerSituation <ead2174@columbia.edu>
Date:   Wed Dec 6 19:23:59 2017 -0500

    fixed triangle

commit 675cc7b788f435522f6fabd7ad28a713b7c98e39
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Wed Dec 6 19:02:36 2017 -0500

    fix fpsmanager segfault

commit 2f6a1fd8d02972dafa0b5e3f64c07b8ea8a8c92b
Merge: 1757a5a 399abea
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Wed Dec 6 18:59:03 2017 -0500

    Merge branch 'testing'

commit 399abea536f519e72ec089a530a9287da5c556ba
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Wed Dec 6 18:54:16 2017 -0500

    lrm update framerate description

commit 1757a5aceb52ffa5c8011817ba0baec3eb602bd7
Author: DyerSituation <ead2174@columbia.edu>
Date:   Wed Dec 6 18:51:22 2017 -0500

    started manual testing

commit fd8d8bdb3d40081dbf793d3035b5ad11baf503d9
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Wed Dec 6 18:29:08 2017 -0500

    add framerate functions

commit 99996105339416135c8255b22cfe3d4db8770616
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Mon Dec 4 19:03:13 2017 -0500

    update main return
```

```
commit 653e6941215fdeb648e368ce5c0a4020dcdda13c
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Mon Dec 4 18:49:13 2017 -0500

    Added implicit return statement to end of main function

commit 6b9b9bbea2420a1ff29c02fc43c1d074c56d5a96
Merge: 31bfe03 34154b1
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Mon Dec 4 18:13:26 2017 -0500

    Merge branch 'master' of https://github.com/bawejakunal/sol

commit 31bfe03867272515b3185d6082da81fb153bed48
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Mon Dec 4 18:13:14 2017 -0500

    Changed to fixed-length array implementation throughout; fixed
       error with return statements

commit 34154b115f3a2c8390696a511d16bac834790f9d
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sun Dec 3 16:05:50 2017 -0500

    drawPoint drawCurve

commit b4b2547d91c34c608edab1f13db254cadd3dfc2f
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sun Dec 3 12:44:48 2017 -0500

    reorder sdl install

commit c22cfa8f3db5cc6db82e143fcce390c7d9f844e9
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Thu Nov 30 18:40:52 2017 -0500

    update sdl gfx install script

commit 66cd0ec1b108abcb6af2cd4654dec59503ae960e
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Thu Nov 30 16:14:34 2017 -0500

    Added SAST file

commit 60d8f20b30c2b22a8648d881ede7da455d2feed3
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Wed Nov 29 19:00:14 2017 -0500
```

```
    Fixed error where predicates could not be boolean expressions

commit 8f26613eebe39d46c4cec1554186280dfea9e99c
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Wed Nov 29 17:36:28 2017 -0500

    Fixed array access issues (yet to fix out-of-bounds checking)

commit a97c0cc4404884e3432fd69aae9f38c8dd319bf8
Merge: 034d044 15eb24d
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Mon Nov 27 19:33:27 2017 -0500

    Merge branch 'master' of https://github.com/bawejakunal/sol

commit 034d04468dba715f8fd84ea16bac4fdeb6305c8f
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Mon Nov 27 19:32:53 2017 -0500

    Added in SAST, scoping, initial array implementation (not
       tested yet)

commit 15eb24ddf46dbffb16480200370b44a49e187d9d
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Thu Nov 16 18:52:04 2017 -0500

    dependency added

commit e91c96b83f3a2fc56513be7eac08d7fb165bcc8b
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Thu Nov 16 18:51:04 2017 -0500

    libsdl dependency broken

commit d662807ad9f6caa90c3ef12886c3f6a1a4fa6a56
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Thu Nov 16 18:45:31 2017 -0500

    add sdl dependencies

commit c7cd2c8e769196c17ea9e4e6c50c88868f74d8ac
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Thu Nov 16 18:24:18 2017 -0500

    update sdl test scripts
```

```
commit 20865361f35473a78133283865ee3e14c2bd7fe7
Merge: 3b8e223 ccf97ae
Author: Kunal Baweja <bawejakunal@users.noreply.github.com>
Date:   Thu Nov 16 18:22:25 2017 -0500

    merge tests into master

    merge tests into master

commit ccf97aed43fc4151e218739266294bb7f38dc663
Author: DyerSituation <ead2174@columbia.edu>
Date:   Thu Nov 16 17:45:00 2017 -0500

    added tests back

commit 3b8e2231142f9a9820e060131d5304bed76c0b31
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Mon Nov 6 18:21:49 2017 -0500

    Added initial creation of SDL window when main() is called

commit 7bfe016f95ab23beda9a86e480119e4c36243e10
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Mon Nov 6 17:38:47 2017 -0500

    precedence tests

commit 31d1131a253e213f55fe958c6c80a03b77804b9b
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Mon Nov 6 17:00:47 2017 -0500

    integration tests

commit de46f50b59c158e6902ee0de1d22f69ab4d3e0ea
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Nov 4 20:25:19 2017 -0400

    add regression tests

commit cb162637e9e7838724e04ac6ed5d7687ce323e13
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Nov 4 19:28:45 2017 -0400

    update master branch with test suite

commit 04bd57104270adc0d137585550fedd932fd1df24
Merge: 15592f8 8b66487
```

```
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Nov 4 19:27:32 2017 -0400

    Merge branch 'testing'

commit 15592f8426e0ab27d710ee294add03bc708134c4
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Nov 4 19:25:57 2017 -0400

    update master travis.yml

commit 8b66487d4c722f2dfacdc92ef7cb5de86df60510
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Nov 4 19:19:51 2017 -0400

    resolve conflict .travis.yml

commit 6c992f5cb41d22ef051efa8da2b4676fa1f82454
Merge: 48c75b4 3baf866
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Nov 4 19:17:54 2017 -0400

    test hello world

commit 3baf866430e6769c85b6b317f993e7b24edc66c3
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Nov 4 19:04:05 2017 -0400

    hello world test

commit b9e88a4df6edea077595f6a6f2d640b2479c4a00
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Nov 4 19:03:35 2017 -0400

    hello world test

commit 840e7769dc0a8f034d743c55e65e51f65281e276
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Sat Nov 4 16:40:24 2017 -0400

    Corrected error with extra escaping for strings in scanner

commit 48c75b46c9c7e116c19883f2b9b25b3233558c25
Merge: a0acaea eff294f
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Nov 4 16:35:53 2017 -0400
```

```
    Merge branch 'master' into testing

commit eff294f72e3a247a93b21ad29de72772ca0635cb
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Sat Nov 4 16:31:44 2017 -0400

    Removed quotes from string literal in scanner

commit a0acaea2054c8e3582d356b79314ab9e0bee1873
Merge: f9d97f9 6ac41bf
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Nov 4 16:21:05 2017 -0400

    Merge branch 'master' into testing

commit 6ac41bfb7603b297adfe1778ed2775e1c68050fa
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Sat Nov 4 16:19:00 2017 -0400

    Changed definition of string literal in codegen; Corrected
       regular expression for string literal in scanner

commit f9d97f9b444ead173beab5e068f8f5eda3bcec73
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Nov 4 16:05:17 2017 -0400

    gold tests

commit 08af5c66e37428d2cb5fe4fdc7dd1a89051e9a31
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Nov 4 15:50:02 2017 -0400

    reorganize travis config

commit 0bdd9c8edc7bd23b43bba7184a48043bea232aed
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Nov 4 14:40:16 2017 -0400

    codegen gives llvm ir

commit a84bb704780fe5fe5db92d997001c2dac7916518
Merge: c2ead10 57a7840
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Nov 4 14:38:53 2017 -0400

    merge codegen.ml for testing
```

```
commit 57a78404249d0bd95498d45e2ca3370398d26ce8
Author: Aditya <aditya.moorthy90@gmailcom>
Date:   Fri Nov 3 18:10:26 2017 -0400

    Completed codegen and added initial sol.ml; yet to finish
       arrays

commit c2ead10f9835930828e49c12f0969eae46ea0327
Merge: 9c00043 deec15d
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Fri Nov 3 00:27:54 2017 -0400

    Merge branch 'master' into testing

commit 9c00043711b37b9efd7e0a8f6890bf3edca5e2a7
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Fri Nov 3 00:23:37 2017 -0400

    install llvm

commit deec15dd18ea96891e7447e0b03b1f02a827a51e
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Thu Nov 2 18:52:07 2017 -0400

    Added initial codegen file - yet to add arrays

commit 05ac9bc3fa5c4801f1c193e585af564ec6ac0c4f
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Thu Nov 2 17:43:00 2017 -0400

    Changed type in Array to accept an expression as the size;
       Added semantic checking to make sure this expression
       evaluates to type Int

commit 11dfdaef5571133e3a4b02dc34347eb10832c0c8
Merge: 2470cbd ae6faa0
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Wed Nov 1 13:36:04 2017 -0400

    Merge branch 'master' into testing

commit ae6faa09a611916e5fe768fd400a79bf1ad9ccab
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Wed Nov 1 12:57:13 2017 -0400

    fix script name
```

```
commit 8d6184b7f818d26420ab1ec3b67509114d7eb57a
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Wed Nov 1 12:53:41 2017 -0400

    fix travis build commands

commit 37b11326659a5657c3a74bc3a381c8920a693f39
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Tue Oct 31 21:52:17 2017 -0400

    build semant

commit c2f10044e15ae63659d623c079992ca85bb21962
Author: Aditya Moorthy <aditya.moorthy90@gmail.com>
Date:   Tue Oct 31 17:43:14 2017 -0400

    Added special assignment case between formals and actuals for a
        function call for arrays

commit 7601a23b81feb2d1ee0bded861cd4ff625f69831
Author: Aditya Moorthy <aditya.moorthy90@gmail.com>
Date:   Tue Oct 31 17:12:06 2017 -0400

    Removed scanner file inside folder

commit bfdaeca67fc9244d2dda0d40a2117996d4dec1c5
Author: Aditya Moorthy <aditya.moorthy90@gmail.com>
Date:   Tue Oct 31 17:11:32 2017 -0400

    Added code vertically for Arrays

commit 2470cbd335e25599624795d326d8e2f58d02a105
Author: DyerSituation <ead2174@columbia.edu>
Date:   Sat Oct 28 21:27:19 2017 -0400

    Added script to build image

commit cc8fb7f82fd4cd3cca3c0a31834532e63218f2b3
Author: DyerSituation <ead2174@columbia.edu>
Date:   Sat Oct 28 21:26:00 2017 -0400

    Added dockerfile used to run llvm 5.0

commit cc687edb8b564bc79ea599d6c94cfc96bb6126da
Merge: fa9ee75 7218ae8
Author: DyerSituation <ead2174@columbia.edu>
Date:   Sat Oct 28 21:24:16 2017 -0400
```

```
    Merge branch 'testing' of https://github.com/bawejakunal/sol
       into testing

commit fa9ee754c449a2bbc1109fca1a0aa387ac5c7042
Author: DyerSituation <ead2174@columbia.edu>
Date:   Sat Oct 28 21:23:44 2017 -0400

    added general testing observations

commit 974b8f6d155f6d64ee2135027605a33196b86dbd
Merge: cc8b52f 7218ae8
Author: Kunal Baweja <bawejakunal@users.noreply.github.com>
Date:   Sat Oct 28 19:53:37 2017 -0400

    Merge pull request #6 from bawejakunal/testing

    Testing

commit 7218ae8c053bb1f6068c9a947788c08191dacd1e
Merge: e2df142 cc8b52f
Author: Kunal Baweja <bawejakunal@users.noreply.github.com>
Date:   Sat Oct 28 19:46:44 2017 -0400

    Merge branch 'master' into testing

commit e2df142331dc8c6c03739aa0d7595c8ea1ba315e
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Oct 28 19:38:54 2017 -0400

    yes flag llvm install script

commit 68a7311265c1071385d6c15ebf666bcd40c7dfc9
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Oct 28 19:17:58 2017 -0400

    try llvm on travis

commit 1ae6a53afd344d240a2c32af2ceedaf8e0e2967c
Author: DyerSituation <ead2174@columbia.edu>
Date:   Sat Oct 28 15:02:16 2017 -0400

    Wrote most test in SOL

commit cc8b52ffdb3f40e40847ada734ea3d2e65023aeb
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Oct 28 14:50:30 2017 -0400
```

```
    parser build statement

commit 4bd2197c3bc0a84072bef5877310eaf5ffc354c8
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Oct 28 14:43:20 2017 -0400

    minimal makefile

commit 8933dd3534360f7e0d44f1a7e1f77b07dcf99d1f
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Oct 28 14:18:44 2017 -0400

    mention 0/1 for conditional statements

commit 61c3d876bcd7351f3afa150679560034f9568df0
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Oct 28 13:59:27 2017 -0400

    fix english grammar

commit e72bc7b7b855ca5ae38c0a69e1fd3256ce4986bf
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Oct 28 13:54:21 2017 -0400

    define string type; update typecasting

commit ddd47a6b5b65adf59a6c656a15d0713641fb8379
Merge: 7145e4d 4d4b0e3
Author: Aditya Moorthy <aditya.moorthy90@gmail.com>
Date:   Fri Oct 27 18:32:30 2017 -0400

    Merge branch 'master' of https://github.com/bawejakunal/sol

commit 7145e4d21c6a19e7a9e84a84b6983c4d78eafcdb
Author: Aditya Moorthy <aditya.moorthy90@gmail.com>
Date:   Fri Oct 27 18:32:02 2017 -0400

    Added floats/char/modulo through all levels; started working on
        arrays (including strings as char arrays)

commit 4d4b0e3b15f189811bc0e6048ebb27227b83b844
Merge: 8e90dd2 aaaf61d
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sun Oct 22 20:06:08 2017 -0400

    Merge branch 'master' of github.com:bawejakunal/sol
```

```
commit 8e90dd287f26a08be45600d03ca808241e1013ee
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sun Oct 22 20:06:00 2017 -0400

    disable travis mail notifications

commit aaaf61d20ea83eae0ef64b6ad5300788ae81858d
Merge: fabf23c 4ed5751
Author: Aditya Moorthy <aditya.moorthy90@gmail.com>
Date:   Sat Oct 21 15:50:41 2017 -0400

    Merge branch 'master' of https://github.com/bawejakunal/sol

commit fabf23ca01169043283cc0be9e0e0c92e4be874a
Author: Aditya Moorthy <aditya.moorthy90@gmail.com>
Date:   Sat Oct 21 15:50:24 2017 -0400

    Initial commit of parser

commit 4ed57516059546908d65ad904ef24b667c035e48
Merge: 72f90bf 3e7a6b2
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Oct 21 15:43:10 2017 -0400

    Merge branch 'scanner'

commit 3e7a6b20895ea2c8ce1872dd825e0481d444b281
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Oct 21 15:41:52 2017 -0400

    readme build status

commit 7d91c8c0d118d3fb7881745578666ca906b1f3bb
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Oct 21 15:39:55 2017 -0400

    travis ocaml setup

commit f0655e581e3fab454612e754a33cf2ff6094ee55
Author: Aditya Moorthy <aditya.moorthy90@gmail.com>
Date:   Sat Oct 21 15:31:26 2017 -0400

    Incremental additions to AST Scanner

commit 72f90bf1c0d1ab9c5cd02bf76faf271e59e310b9
Author: Aditya Moorthy <aditya.moorthy90@gmail.com>
```

```
Date:   Sat Oct 21 13:57:24 2017 -0400

    Updated Scanner; Added unary - to AST

commit d9326b37fb76aabf11a8a549171979213925543f5
Merge: e2e58e2 8b7ca18
Author: DyerSituation <ead2174@columbia.edu>
Date:   Sat Oct 21 13:23:16 2017 -0400

    Merge branch 'ast' into scanner

commit e2e58e2d664bbb05c6fee5c9356b0958cfee7c94
Merge: 97bcfc1 538efac
Author: DyerSituation <ead2174@columbia.edu>
Date:   Sat Oct 21 13:22:32 2017 -0400

    Merge branch 'scanner' of https://github.com/bawejakunal/sol
       into scanner

commit 538efacdf3d366051b48f6fc1f723da0eacc6859
Author: Aditya Moorthy <aditya.moorthy90@gmail.com>
Date:   Sun Oct 15 20:39:55 2017 -0400

    Corrected symbols in operator tables

commit 9c1bd2b6804f7d8b6fd889958e2140c93bebca9e
Author: Aditya Moorthy <aditya.moorthy90@gmail.com>
Date:   Sun Oct 15 11:37:07 2017 -0400

    LRM second draft

commit 20301436b17b23710a13d18882fc90e5d86cf19b
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sun Oct 15 00:29:33 2017 -0400

    lrm first draft

commit ba2e452491cf2bc93c85a3d7135972873606826e
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Oct 14 23:58:43 2017 -0400

    lrm: 1 section left; urgent review required

commit 74540315c93c4b0efc62a40524982817d55aab06
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Sat Oct 14 22:07:54 2017 -0400
```

```
    lrm sec 5

commit 64cb91e9a639b2a539267a055aafc68332100433
Author: Aditya Moorthy <aditya.moorthy90@gmail.com>
Date:   Thu Oct 12 21:14:36 2017 -0400

    Added to LRM until section Declarations; re-arranged
       Typecasting to come under Expressions and Operators rather
       than Lexical Conventions

commit b16a2bf5d007685f475377b9fb5b03e660a75125
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Thu Oct 12 18:53:59 2017 -0400

    broken arithmetic operators section

commit 8b7ca18ea20da22d3114343bed079793b18da1b6
Merge: 97bcfc1 78ba1ec
Author: DyerSituation <ead2174@columbia.edu>
Date:   Thu Oct 12 18:42:59 2017 -0400

    Merge branch 'ast' of https://github.com/bawejakunal/sol into
       ast

commit 78ba1ecea4d79e4adaeba093578ba51bda71073a
Author: Gergana Alteva <glalteva@gmail.com>
Date:   Thu Oct 12 18:30:03 2017 -0400

    began ast

commit 97290f36ab21616109442849cd4d72dbe2fe0bf0
Author: Aditya Moorthy <aditya.moorthy90@gmail.com>
Date:   Thu Oct 12 17:15:31 2017 -0400

    Updated escape sequences

commit fdc700cf48ff8e3038067172d4a1384ae5ffcfd8
Author: Aditya Moorthy <aditya.moorthy90@gmail.com>
Date:   Thu Oct 12 16:57:35 2017 -0400

    Updated tokens in scanner

commit 97bcfc1288c37efd2120c12a4bdabe24fe2ff67a
Author: Aditya Moorthy <aditya.moorthy90@gmail.com>
Date:   Thu Oct 12 01:07:53 2017 -0400

    Sample SDL program
```

```
commit 899ce46a75aa2a1ac36ff94e130090db34beaf2b
Author: Gergana Alteva <glalteva@gmail.com>
Date:   Wed Oct 11 19:52:27 2017 -0400

    started ast

commit a3339a2b5f1653874b2ed8dd2bb561af762a948e
Author: Aditya Moorthy <aditya.moorthy90@gmail.com>
Date:   Thu Sep 28 18:34:56 2017 -0400

    Initial scanner

commit 18f92481541f13793c4ce8307e1076ba8f5b3a63
Author: Aditya Moorthy <aditya.moorthy90@gmail.com>
Date:   Sat Sep 23 13:28:10 2017 -0400

    Added wordwrap in code sample

commit c7d68aeb6acf679d5f8ed57b52c60b00f685f3f7
Author: Aditya Moorthy <aditya.moorthy90@gmail.com>
Date:   Sat Sep 23 13:23:52 2017 -0400

    Fixed all indentations in code sample

commit a093da09a1c94924538c05c1d43bf7050c3b513a
Author: Aditya Moorthy <aditya.moorthy90@gmail.com>
Date:   Sat Sep 23 13:18:18 2017 -0400

    Formatted code sample; removed full stops after bullet points;
       corrected accents on 'e'

commit 16ce7395024a81e54566c6ed5ddc5a573c434325
Author: Kunal Baweja <bawejakunal15@gmail.com>
Date:   Thu Sep 21 18:47:59 2017 -0400

    proposal latex draft

commit 3dca6dc26cf942d12c602119b5a9642dc68024b7
Author: Kunal Baweja <bawejakunal@users.noreply.github.com>
Date:   Thu Sep 21 17:15:14 2017 -0400

    add text gitignores

commit cc7e89043f8b0bdd4d3551333f351e904a86a7d7
Author: Kunal Baweja <bawejakunal@users.noreply.github.com>
Date:   Tue Sep 12 13:04:04 2017 -0400
```

```
    Initial commit
```