bugsy

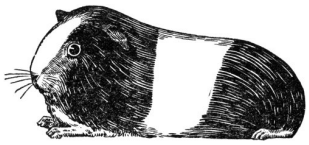
Final Report

PLT Spring 2021

Ben Snyder Sofía Sánchez Zárate Evan Tilley

Michael Winitch Jason Cardinale

April 26, 2021



Contents

1 Introduction

Bugsy is a statically typed programming language focused on creating 2D graphics and animations. This language is loosely inspired by p5.js and Processing.The syntax is mainly Java style. This language has built in shape functions allowing users to create shapes on a canvas. Animations can also be run on the shapes such as scaling or rotating. We want this language to be of assistance to those who want to work with visualization, but may not necessarily have a strong programming background. This language uses OpenGL to create the visualizations.

1.1 White Paper

Bugsy is a language designed to make it easy for the programmer to write openGL graphical programs. Generate animations, and static images with ease by

2 Tutorial

2.1 Environment Setup

Clone the source code from GitHub:

```
$ git clone https://github.com/BenjaminSnyder/bugsy.git
```

Navigate to the bugsy folder.

```
$ cd bugsy
```

You can either use the provided dockerfile or ensure that you have the requisite packages. The requite ubuntu/debian packages are: ocaml opam menhir llvm-10 llvm-10-dev libglew-dev freeglut3-dev
You should run the command below to ensure that your environment is set up correctly.

```
$ eval $(opam env)
```

2.2 Compilation Guide

From here, run make to build the compiler

```
$ make
```

To ensure that everything is working properly, run the test script. If all tests pass, your environment is set up correctly.

```
$ ./scripts/testall.sh
```

Given these preconditions, you can then compile your source code into an executable:

```
$ ./scripts/bugsyc <filename>
```

If for some reason you only wanted to generate LLVM IR, you can run

```
$ ./bugsy.native <filename>
```

2.3 Language Tutorial

To begin, let us make a starter program that draws a square on a canvas. Let us call this file square.bug (the bugsy language uses .bug file extensions).

```
num main() {
    num cx;
    num cy;
    num x;
    num y;
    num cw;
    num ch;
    num s;
    cx = 0;
    cy = 0;
    cw = 1500;
    ch = 1000;
    x = cw / 2;
    y = ch / 2;
    s = 75;
    string stroke;
    string sq_fill;
    stroke = "0.0 0.3 0.5";
    sq_fill = "0.9 0.9 0.2";
    num thickness;
    thickness = 5;
    string id;
    id = "";
    init_canvas();
    add_square(x, y+300, s, stroke, thickness, sq_fill, id);
    add_canvas(cw, ch, cx, cy);
    return 0;
}
```

Any Bugsy program must have a main method of type num. We first have to declare our attributes, such as the size of the canvas. Then we assign all

of these variables to values. Note that the fill attribute (sq_fill) is actually a string. To actually begin drawing, we first call init_canvas(). Then we can call the built in Bugsy function add_square and pass in the parameters to say where the shape will go along with its colors. We then have to end with add_canvas once we are done with all of our drawing. Finally we return 0 in our main function. To actually run this script, make sure you run make first. Then, run the following:

```
$ ./scripts/bugsyc square.bug
```

Then to actually run this executable:

```
$ DEBUG=0 ./square
```

A window shows pop up with a square. You have to pass in the DEBUG=0 flag so that the window does not close instantly.

3 Language Reference Manual

3.1 Comments

Comments in bugsy are used to document code and are ignored by the parser, written solely for readability purposes. Single Line Comments Single line comments in bugsy are creating using two forward slashes.

```
// This is an example of a single line comment
```

Multi-line Comments bugsy also supports multi-line comments. To start a multi-line comment type /* and close it with */

```
/*
    This is an example
    of a multi-line comment
*/
```

3.2 Reserved Words

bugsy has words that are reserved for specific uses and thus cannot be used in a typical manner. The following is a list of reserved words that can not be used for the names of variables, classes, functions:

- if
- else
- while
- for
- True
- true
- False
- false
- not
- num
- bool
- string
- array
- and
- or
- void

3.3 Data Types

bugsy has a variety of data types, each suited for representing different types of information.
Primitive Data Types **num** is used to express any number, including both integers and floating point numbers.

```
num a;
a = 10;
num b;
b = 10;
```

bool is used for boolean truth values (either true or false). As a note the word true or false can be capitalized, but it is not required.

```
bool x;
x = True;
bool y;
y = true;
bool z;
z = false;
```

The string type is used to represent a concatenation of characters. Strings are made from ASCII characters and are declared using double quotes.

```
string greet;
greet = "Hello world!";
string code;
code = "abc 123";
```

Built in Shapes bugsy is an animation language and has pre-defined shape objects. These include typical shapes like squares and circles, but also user defined polygons as well. Below is a table of all built in shapes.

Function	Description	Parameters	Example
`add_square -> string`	Defined from an origin x, y and given a num size s. As with all shape functions, it is passed stroke and fill color strings and stroke thickness.	`add_square(x, y, s, stroke, thickness, fill, id);`	`add_square(0, 0, 50, "0.5 0.5 0.5", 2, "0.2 0.2 0.2", "");`
`add_rectangle -> string`	Defined from an origin point and takes an input width num and height num.	`add_rectangle(x, y, w, h, stroke, thickness, fill, id);`	`add_rectangle(0, 0, 100, 200, "0.5 0.5 0.5", 2, "0.2 0.2 0.2", "");`
`add_triangle -> string`	3-pointed polygon defined from an origin point, with num height and width inputs.	`add_triangle(x, y, b, h, stroke, thickness, fill, id);`	`add_triangle(0, 0, 100, 200, "0.5 0.5 0.5", 2, "0.2 0.2 0.2", "");`
`add_circle -> string`	Infinite-sided polygon defined from an origin point, with a num radius input.	`add_circle(x, y, r, stroke, thickness, fill, id);`	`add_circle(0, 0, 200, "0.5 0.5 0.5", 2, "0.2 0.2 0.2", "");`
`add_ellipse -> string`	Infinite-sided polygon defined from an origin point, and a num w and h which are the which are the horizontal and vertical radii inputs, respectively.	`add_ellipse(x, y, w, h, stroke, thickness, fill, id);`	`add_ellipse(0, 0, 100, 200, "0.5 0.5 0.5", 2, "0.2 0.2 0.2", "");`
`add_regagon -> string`	Creates a regular x sided polygon of radius r at origin point pt, where n is a num input and x, y is the origin point.	`add_regagon(x, y, n, r, stroke, thickness, fill, id);`	`add_regagon(0, 0, 10, 200, "0.5 0.5 0.5", 2, "0.2 0.2 0.2", "");`
`add_canvas -> void`	Defines a 2D plane whose boundaries constrain the locations of shapes. Has width and height parameters of type num.	`add_canvas(w, h, x, y);`	`add_canvas(1000, 500, 0, 0);`

3.4 Variables

In bugsy variables are declared using a type and a name. The type comes first followed by a variable name and then an equals sign. The right hand side of the expression is returned as a value. If the variable is only being declared, then a semicolon follows the name instead of an equals sign.
The grammar is as follows:

type ID equals expression

type ID

```
num a;
a = 10;
num b;
b = 20;
num a = b = 4 * 5;
string hello = "hello sirs.";
```

Variable Naming Variables names follow the following regex `[a-zA-z_][a-zA-Z0-9_]*` and as stated in the Reserved Words subsection, cannot be a reserved word.

3.5 Arrays

An array is a collection of nums that can be accessed with indices. Declaring Arrays Arrays in bugsy are declared by describing the type (num) followed by square brackets containing the size of the array (`[10]`) followed by a variable name.
The grammar goes:

type LeftBracket number RightBracket ID

type LeftBracket number RightBracket ID equals LeftBracket li_contents Right Bracket

With `li_contents` being the contents of the array (name coming from list contents).

```
num[10] myArray;
```

Initializing Arrays Arrays in bugsy must be declared and later initialized. Once declared, the value at a certain index in the array can be modified by using the name of the array, followed by square brackets containing the index that you want to reference or change.

```
num[3] myFirstArray; //declared
myFirstArray = [5.4, 4.3, 6.3];
num[2] myThirdArray; //declared without initialization (defaults all
    values to 0)

//here we initialize the values inside the array
myThirdArr[0] = 7; //this notation allows you to directly assign numbers
myThirdArr[1] = 8; //to the index between the square brackets
```

3.6 Operators and Arithmetic

All arithmetic done in bugsy will be expressions composed of operators and values. bugsy supports both unary and binary operators.

The grammar for expressions breaks down into literals, booleans, unary operations, and binary operations. The unary and binary operations either include one or two more *expr's* in the pattern matching so that it can expand. For instance, addition is done with *expr* PLUS *expr*.
Unary Operators bugsy utilizes three four of unary operators, increment (`++`), decrement (`--`), the unary minus sign (`-`) in front of an expression to flip the sign of a numerical value (or a variable containing a numerical value), and an exclamation mark (`!`) in front of an expression to flip the resulting boolean. Using the unary minus on a boolean expression is not valid.

Grammar:
OPERATOR expr
expr OPERATOR

Operator	**Character(s)**	**Example**

Increment	`++`	`num i;`
		`i = 0;`
		`i++;`
		`++i;`

Decrement	`--`	`num j;`
		`j = 0;`
		`j--;`
		`--j;`

Unary minus (negation operator)	`-`	`num x;`
		`x = -5;`
		`num y;`
		`y = -5;`

Not	`!`	`bool x;`
		`x = true;`
		`bool y;`
		`y = !x;`

Binary Operators The binary operators in bugsy are outlined as follows in the table below. Addition (`+`), subtraction (`-`), multiplication (`*`), division (`/`), plus equals (`+=`), minus equals (`-=`), multiplication equals (`*=`), division equals (`/=`), assignment (`=`), modulus (`%`), and exponentiation (`^`) all result in numerical values and can only take expressions that produce numerical val-

ues. The greater than comparison (`>`), less than comparison (`<`), greater than or equal to comparison (`>=`), less than or equal to comparison (`<=`), equality check (`=?`), and not equals to (`!=`) result in booleans but only accept numeric values. And (`and`), and or (`or`) also produce booleans only accept boolean expression inputs. The exponent operator will not flip bits.

Grammar: *expr OPERATOR expr*

Note that in the following examples, x and y are initialized as

```
num x;
x = 0;
num y;
y = 0;
```

Operator	**Character(s)**	**Example**
Addition	`+`	`x + y;`
Subtraction	`-`	`x - y;`
Multiplication	`*`	`x * y;`
Division	`!`	`x / y;`
Assignment	`=`	`x = y;`
Not equals (neq)	`!=`	`x != y;`
Greater than	`>`	`x > y;`
Less than	`<`	`x < y;`
Greater than or equal to	`>=`	`x >= y;`
Less than or equal to	`<=`	`x <= y;`

Logical Operators As previously mentioned, not (`!`), and (`'and'`), or (`'or'`) are bugsy's logical operators. Booleans can be expressed as either (`1`), any positive number, (`'true'`), or (`'True'`) for `true`. and (`0`), any negative number, (`'false'`), or (`'False'`) for `false`.

The grammar is the same as the previous operators, depending on if it's unary or binary.

3.7 Control Flow

bugsy supports a majority of the most common aspects of control flow. As an animation oriented language, looping plays an important role in displaying the intended content and will be used within several of the standard library functions. Loops
While Loops while loops in bugsy work identical to a while loop in other imperative languages such as Python, C, or Java. A while loop is uniquely defined by its two major components which are the conditional statement which bugsy requires be enclosed in parentheses following the `while` keyword. After the close parentheses the code intended to be performed each iteration of the loop will be enclosed in curly braces.
A while loop will continue to execute the code encapsulated within the opening and closing curly braces until the condition defined within the parentheses evaluates to false (Refer to subsection 4.1 **Primitive Data Types** and subsection 8.3 **Logical Operators**).
Grammar:

while LeftParen expr RightParen LeftBracket stmt_list RightBracket

```
num val;
val = 0;
while(True) {
  // execute code
  val = val + 1;
}
```

For Loops for loops, similar to while loops will execute a block of code enclosed in curly braces until the end condition is met. The for loop is defined by three statements enclosed in parentheses following the keyword `for`.
The first statement declares a num variable and initializes to some starting value. The second statement, separated by a semicolon, defines the comparison condition involving the previously declared num and another num. The third statement provides the increment or decrement step sizes for the num declared in the first statement.
Grammar:

for LeftParen Num ID SEMI ID CMP_OPERATOR SEMI ID INC_OPERATOR RightParen LeftBracket stmt_list RightBracket

```
num val;
num i;
i = 0;
val = 0;
for(i = 0; i < 10; i++) {
  val = val + i;
```

```
}
```

Conditional Statements The `if` statement will execute a block of code enclosed in curly braces if the provided condition is satisfied. The condition follows the keyword if and is enclosed in parentheses. The `else` statement is used as a catch-all which will execute if none of the previously defined if conditions are met. The else statement consists of the keyword else followed by curly braces as it does not have its own condition.
Grammar: if LeftParen expr RightParen LeftBracket stmt_list RightBracket

if LeftParen expr RightParen LeftBracket stmt_list RightBracket else LeftBracket stmt_list RightBracket

if LeftParen expr RightParen LeftBracket stmt_list RightBracket elif LeftParen expr RightParen LeftBracket stmt_list RightBracket

if LeftParen expr RightParen LeftBracket stmt_list RightBracket elif LeftParen expr RightParen LeftBracket stmt_list RightBracket else LeftBracket stmt_list RightBracket
I . . . else

```
num val;
val = 5;
num res;
res = 0;
if(val =? 5) {
  res = 1;
} else {
    res = 3;
}
```

3.8 Functions

Functions in bugsy are designed to be simple, clean, and reusable. The syntax for defining a function is as follows.

```
return_type function_name(input1_type input1, input2_type input2...)
    {...}
```

Where the types of the return value and the parameter can be any primitive or custom type. A function that does not return anything can be specified with the return type void. The following is an example of a function which finds the greatest common denominator of two numbers.

```
num gcd(num a, num b) {
   while (a != b) {
      if (a > b) {
         a = a - b;
} else {
```

```
  b = b - a;
}
  }
  return a;
}
```

Also note that parameters are passed by value and thus the return value of a function must be set equal to a variable.
Grammar:
V type ID LeftParen formal_list RightParen LeftBracket var_list stmt_list RightBracket

3.9 Memory Management

In bugsy, memory is leaked by default.

3.10 Standard Library

The standard library in bugsy consists of a variety of functions which are primarily used for animation and graphical manipulation. Below are the functions included in the standard library.
moveById
Definition
```
moveById(string id, num translateX, num translateY, num time)
```
Description
moveById will use the passed in parameters to call a function defined on the backend which determines which shape the id belongs to, and changes its position according to the translateX and translateY variables. The transformation will take time seconds.
Example Usage
```
id = add_circle(0, 0, 50, "0.5 0.5 0.5", 2, "0.2 0.2 0.2", "");
moveById(id, -2000, -2000, 1);
```
scaleById Definition
```
scaleById(string id, num scale, num time)
```
Description Scales the shape identified by string by scale in time. Example Usage
```
id = add_circle(0, 0, 50, "0.5 0.5 0.5", 2, "0.2 0.2 0.2", "");
scaleById(id, 3, 1);
```
rotateById Definition
```
rotateById(string id, num angle, num time)
```
Description Rotates the shape identified by string by scale in time.
Example Usage
```
id = add_circle(0, 0, 50, "0.5 0.5 0.5", 2, "0.2 0.2 0.2", "");
rotateById(id, 360, 1);
```

4 Project Plan

4.1 Planning Development and Testing

4.2 Style Guide

Code written between the “let” and “in” keywords is tabbed over 2 spaces.
try to be consistent within a file on snake_case vs camelCase

4.3 Project Timeline

4.3.1 Februrary 3rd: Proposal

January 20: Pick a project
January 24: Start proposal
January 28: Finalize proposal

4.3.2 February 24: LRM Parser

February 14: Start LRM
February 16: Start parser, scanner, AST
February 22: Finalize LRM
February 24: Finalize parser, scanner (for first milestone)

4.3.3 Hello World (simple animation)

March 11: Set deadlines, meet with Hans
March 14: Code sesh! Finish parser, scanner, AST !!
March 18: Do codegen.ml and connect to OpenGL
March 19-21: Finish anything remaining before the 24th

4.3.4 Mid-April

April 3: Start OpenGL library
April 10: Start semantic checking of classes and arrays, work on variable access
April 17: Finish implementing OpenGL library

4.3.5 April 26: Project Due

April 20: Make sure codegen works, finalize semantic checking, start final report presentation material
April 23: Finish demo implementation/take video
April 25: Finalize testing details, finish presentation slides
April 26: clean up the code base for submission and write report

4.4 Team Roles

Manager: Sofía Sánchez-Zárate
Language Guru: Jason Cardinale
Tester: Michael Winitch
System Architect: Ben Snyder Evan Tilley

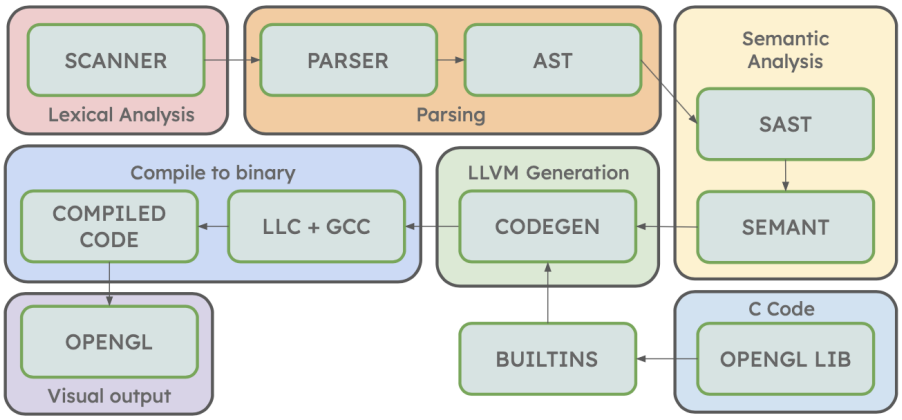
4.5 Dev Environment

Our Development Environment was hosted on a shared linux server. This server was hosted on Google Cloud Platform
We used the following programming and development environments when creating bugsy:
Libraries and Languages: Ocaml Version 4.05.0 including Ocamlyacc and Ocammllex and LLVM Version 10.0.0 was used, additionally, openGL version 4.6, the libraries freeglut3, and libglew-dev.
Software: Development was done in vim, Atom and VS Code.
OS: Development was done on OS X 10.15.7 and Debian 10.
Version Control: Github-Hosted Git Repository

4.6 Project Log

See Appendix

5 Compiler Architecture



5.1 Compiler

The scanner uses the tokens in the parser and reads them in for the parser
The parser takes the scanner's tokens and the types in the ast and builds the AST

Semant takes the AST and traverses it, building a semantically checked syntax tree using types from the SAST
Codegen takes the SAST and builds a set of LLVM IR

5.2 Contributions

OpenGL Library, Built-ins: Jason Sofía
Scanner, Parser, AST: Ben
Parser, Scanner, AST, SAST, Semant: Everyone (Ben Michael for classes, Evan for arrays, Jason Sofía for shapes)
Codegen: Evan, Michael, Ben

6 Testing Plan

To make the test suite more automated we used the Bash script included with MicroC to run through a set of example bugsy files. In test/tests you can see all of the example test files that are either test-*.bug or fail-*.bug. Then all of these files are associated with output files either in test/goldenset or test/fails. The goldenset directory contains desired outputs while fails contains desired error messages. The script that runs all of the files and compares them to an output is in the scripts directory and is called testall.sh.

6.1 Regression Tests

To make the test suite more automated we used the Bash script included with MicroC to run through a set of regression tests. In test/tests you can see all of the example test files that are either test-*.bug or fail-*.bug. Then all of these files are associated with output files either in test/golden_set or test/-fails. The golden_set directory contains desired outputs while fails contains desired error messages. The script that runs all of the files and compares them to an output is in the scripts directory and is called testall.sh.

6.2 Exampe Programs

See Section 8.2 in the appendix.

6.3 Graphical Testing

We also wanted to test our graphics components and needed an easy and automated way to do so. Our solution was to include a print function in the builtins.c file:

```
void printShapes() {
    for(int i = 0; i < count; i++) {
        struct Shape s = shapes[i];
        printf("Shape: %s ID: %s\nx: %d y: %d\nn: %d r: %d\n", s.shape,
            s.shapeId, (int)s.x, (int)s.y, (int)s.n, (int)s.r);
```

```
        printf("w: %d h: %d\nb: %d s: %d\nx1: %d y1: %d\nx2: %d y2:
            %d\n", (int)s.w, (int)s.h, (int)s.b, (int)s.s, (int)s.x1,
            (int)s.y1, (int)s.x2, (int)s.y2);
        printf("stroke: %s thickness: %d\nfill: %s\n", s.stroke, (int)
            s.thickness, s.fill);
    }
}
```

This print function is called twice. It prints when all of the shapes have been added to the canvas. The print function prints a second time when the canvas is done. Even though our language has graphics, we can use this output to compare with an output file to make sure we are correctly initializing our shapes, and for testing our animation functions. The graphics window that is produced will also keep running, which would be undesired for an automated test script. To get around this, we pass in an environment variable called DEBUG which allows the window to automatically close after all shapes have been added and their animations have finished.

```
if(strcmp(getenv("DEBUG"), "1") != 0) {
        glutMainLoop();
    }
```

That means the user running the test script does not manually have to close the window.

7 Lessons Learned

7.1 Michael

Careful planning in the early stages of this project can save you a tremendous amount of trouble during later parts of the project. Also, I think it is very important to set realistic goals, and to not be too ambitious with the language you create.

7.2 Jason

It's important to be able to think of alternative solutions to problems. If one particular way is not working, it can often be a lot more efficient to try to make it work another way rather than spending too much time trying to get an implementation working that may not even be plausible.

7.3 Ben

The most important learning that I took was how hard it can be to get things finished even if there aren't that many lines of code to be added. Try to tackle only one major feature for your language and do it well. If you spread yourselves thin, then the language will have some rough spots.

7.4 Evan

I learned that if you have a problem it's important to identify it at its roots. A lot of issues that seemed inexplicable were resolved by inspecting the LLVM and determining what was going wrong there. It's interesting how understanding code at a lower level can help fix issues at a higher level.

7.5 Sofia

My most important learning is that strong yet understanding/empathetic managing is harder than I expected. When scheduling meetings or creating deadlines, I would be very lenient with everyone's schedule (allowing people to miss when they said they would work on the project at another time, etc.). Everyone in the team still did the tasks they needed to do, but I think it would've been much more beneficial to have more synchronous and more frequent work sessions. As far as the technical side of the project, I realize how complicated it can be to all work on different parts of the same files, even with source control. I worked a lot with Jason, and we managed to get a lot of stuff working on a particular branch which later got updated, breaking a lot of our code. So, I'll definitely be more careful about forking/branching in the future!

8 Appendix

8.1 Source Code

```
**************************************************
Makefile
**************************************************

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

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

.PHONY: all
all : clean bugsy.native builtins.o

.PHONY: bugsy.native
bugsy.native :
   ocamlbuild -I src -use-ocamlfind -pkgs llvm,llvm.analysis -cflags
       -w,+a-4-42 \
     bugsy.native

builtins.o:
   cc -c -o _build/builtins.o src/builtins.c -lm
```

```
# "make clean" removes all generated files

.PHONY : clean
clean :
   ocamlbuild -clean
   rm -rf testall.log *.diff bugsy.native scanner.ml parser.ml parser.mli
   rm -rf _build/builtins.o
   rm -rf *.cmx *.cmi *.cmo *.cmx *.o *.s *.ll *.out *.exe
   rm -rf _build tmp

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

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

bugsy : $(src/OBJS)
   ocamlfind ocamlopt -linkpkg -package llvm -package llvm.analysis
       $(src/OBJS) -o bugsy

scanner.ml : scanner.mll
   ocamllex src/scanner.mll

parser.ml parser.mli : parser.mly
   ocamlyacc src/parser.mly

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

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

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

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

```
# Building the tarball

#TESTS = hello add beans bool

TESTS = add1 arith1 arith2 arith3 array1 array2 circle decrement1
    decrement2 ellipse fib for1 for2 func1 func2 func3 \
   func4 func5 func6 func7 func8 gcd2 gcd global1 global2 global3 \
   hello if1 if2 if3 if4 if5 increment1 increment2 local1 local2 moveBy
       ops1 ops2 rectangle regagon rotateBy scaleBy square triangle
       var1 var2  \
   while1 while2

FAILS = array assign1 assign2 assign3 dead1 dead2 expr1 expr2 for1 for2 \
   for3 for4 for5 func1 func2 func3 func4 func5 func6 func7 func8 \
   func9 global1 global2 if1 if2 if3 nomain return1 return2 while1 \
   while2

TESTFILES = $(TESTS:%=tests/test-%.bug) $(TESTS:%=golden_set/test-%.out)
    \
      $(FAILS:%=tests/fail-%.bug) $(FAILS:%=fails/fail-%.err)

EXAMPLES = fact.bug shape_test.bug \
arrays.bug animation-array-example.bug loop.bug fizzbug.bug


TARFILES = src/ast.ml src/codegen.ml Makefile src/bugsy.ml
    src/parser.mly\
              README src/scanner.mll src/sast.ml src/semant.ml
                  scripts/testall.sh\
              $(TESTFILES:%=test/%) src/builtins.c scripts/bugsyc \
              $(EXAMPLES:%=examples/%) Dockerfile

bugsy.tar.gz : $(TARFILES)
   cd .. && tar czf bugsy.tar.gz \
     $(TARFILES:%=bugsy/%)


**************************************************
ast.ml
**************************************************

(* Abstract Syntax Tree and functions for printing it *)
module StringMap = Map.Make(String)

type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater |
    Geq |
         And | Or | Mod | Pluseq | Mineq | Multeq | Diveq |
         PreInc | PostInc | PreDec | PostDec

type uop = Neg | Not
```

```
type expr =
    NumLit of string
  | StrLit of string
  | BoolLit of bool
  | ArrayLit of expr list
  | IntLiteral of int
  | Id of string
  | Access of string * string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Assign of string * expr
  | Construct of string * expr list
  | ArrayAssign of string * expr * expr
  | ArrayAccess of string * expr
  | Crementop of expr * op
  | Call of string * expr list
  | ClassCall of string * string * expr list
  | Noexpr


type typ = Num | Bool | Void | Int | String | Array of typ * expr |
    Object of classTyp

and bind = typ * string

and classTyp = {
  className : string;
  instanceVars : (typ * expr) StringMap.t;
}


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

type construct_decl = {
  ctformals : bind list;
  ctlocals : bind list;
  ctbody : stmt list;
}

type func_decl = {
    (* make type mutable for modification in codegen *)
    mutable typ : typ;
```

```
    fname : string;
    formals : bind list;
    locals : bind list;
    fbody : stmt list;
}

type cdecl = {
  cname : string;
  cdvars : bind list;
  cdconst: construct_decl list;
  cdfuncs: func_decl list;
}

type program = bind list * func_decl list * cdecl list

(* Pretty-printing functions *)

let string_of_op = function
    Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
  | Mod -> "%"
  | Equal -> "=?"
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "and"
  | Or -> "or"
  | Pluseq -> "+="
  | Mineq -> "-="
  | Multeq -> "*="
  | Diveq -> "/="
  | PreInc | PostInc -> "++"
  | PreDec | PostDec -> "--"

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

let rec string_of_expr = function
    NumLit(nl) -> nl
  | IntLiteral(l) -> string_of_int l
  | StrLit(str) -> "\"" ^ str ^ "\""
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | ArrayLit(el) -> "[" ^ String.concat ", " (List.map (fun e ->
      string_of_expr e) el) ^ "]"
```

```
  | Id(s) -> s
  | Access(c, v) -> c ^ "." ^ v
  | 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
  | Crementop(e, o) -> (match o with
     PreInc -> string_of_op o ^ string_of_expr e
    | PostInc -> string_of_expr e ^ string_of_op o
    | PreDec -> string_of_op o ^ string_of_expr e
    | PostDec -> string_of_expr e ^ string_of_op o
    | _ -> "ERROR")

  | Assign(v, e) -> v ^ " = " ^ string_of_expr e
  | Construct(a, e) -> "new " ^ a ^ "(" ^ String.concat ", " (List.map
      string_of_expr e) ^ ")"
  | ArrayAccess(a, e) -> a ^ "[" ^ string_of_expr e ^ "]"
  | ArrayAssign(a, left, right) -> begin a ^ "[" ^ string_of_expr left ^
      "] = " ^ string_of_expr right; end
  | Call(f, el) ->
     f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | ClassCall(c, f, el) ->
     c ^ "." ^ f ^ "(" ^ String.concat ", " (List.map string_of_expr
         el) ^ ")"
  | Noexpr -> ""

let rec add_level (listy, level) = match listy with
  [] -> []
  | hd::li' -> (hd,level):: add_level (li', level)

let rec string_of_stmt (stmt,level) = match stmt with
    Block(stmts) ->
      "{\n" ^ (String.make level '\t') ^ String.concat (String.make
          level '\t') (List.map string_of_stmt (add_level (stmts,
          level))) ^ (String.make (level-1) '\t') ^ "}\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
  | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")" ^
      string_of_stmt (s, (level))
  | If(e, s1, s2) -> "if (" ^ string_of_expr e ^ ")" ^
     string_of_stmt (s1, (level)) ^ (String.make (level-1) '\t') ^
         "else" ^ string_of_stmt (s2, (level))
  | For(e1, e2, e3, s) -> "for (" ^ string_of_expr e1 ^ " ; " ^
      string_of_expr e2 ^ " ; " ^
     string_of_expr e3 ^ ") " ^ string_of_stmt (s, (level+1))
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt
      (s, (level+1))

let rec string_of_typ = function
    Num -> "num"
  | Bool -> "bool"
```

```
  | Void -> "void"
  | String -> "string"
  | Array(t, e) -> string_of_typ t ^ "[" ^ string_of_expr e ^ "]"
  | Object(clas) -> clas.className
  | Int -> raise (Failure "do not print ints")

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

let string_of_const_decl const_decl =
  "constructor(" ^ String.concat ", " (List.map snd
      const_decl.ctformals) ^
  ") {\n\t\t" ^
  String.concat "\t\t" (List.map string_of_vdecl const_decl.ctlocals) ^
      "\t\t" ^
  String.concat "\t\t" (List.map string_of_stmt (add_level
      (const_decl.ctbody, 1))) ^ "\t" ^
  "}\n"

let string_of_fdecl (fdecl, level) =
  string_of_typ fdecl.typ ^ " " ^
  fdecl.fname ^ "(" ^ String.concat ", " (List.map snd fdecl.formals) ^
  ") {\n" ^ (String.make (level) '\t') ^
  String.concat (String.make level '\t') (List.map string_of_vdecl
      fdecl.locals) ^ (String.make level '\t') ^
  String.concat (String.make level '\t') (List.map string_of_stmt
      (add_level (fdecl.fbody, (level+1)))) ^ (String.make (if level-1
      < 0 then 0 else level-1) '\t') ^
  "}\n"

let string_of_cdecl (cdecl, level) =
  "class " ^ cdecl.cname ^ " {" ^ "\n" ^ (if (List.length cdecl.cdvars)
      < 1 then "" else "\t") ^
  String.concat "\t" (List.map string_of_vdecl cdecl.cdvars) ^ (if
      (List.length cdecl.cdconst) < 1 then "" else "\t") ^
  String.concat "\t" (List.map string_of_const_decl cdecl.cdconst) ^
      "\t" ^
  String.concat "\t" (List.map string_of_fdecl (add_level
      (cdecl.cdfuncs, (level+1)))) ^ (String.make (level-1) '\t') ^
  "}\n"

let string_of_program (vars, funcs, classes) =
  String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
  String.concat "" (List.map string_of_cdecl (add_level (classes, 1))) ^
      "\n" ^
  String.concat "\n" (List.map string_of_fdecl (add_level (funcs, 1)))


**************************************************
bugsy.ml
**************************************************
```

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

type action = Ast | Sast | LLVM_IR | Compile

let () =
  let action = ref Compile in
  let set_action a () = action := a in
  let speclist = [
    ("-a", Arg.Unit (set_action Ast), "Print the AST");
    ("-s", Arg.Unit (set_action Sast), "Print the SAST");
    ("-l", Arg.Unit (set_action LLVM_IR), "Print the generated LLVM IR");
    ("-c", Arg.Unit (set_action Compile),
      "Check and print the generated LLVM IR (default)");
  ] in
  let usage_msg = "usage: ./bugsy.native [-a|-s|-l|-c] [file.mc]" in
  let channel = ref stdin in
  Arg.parse speclist (fun filename -> channel := open_in filename)
      usage_msg;

  let lexbuf = Lexing.from_channel !channel in
  let ast = Parser.program Scanner.token lexbuf in
  match !action with
    Ast -> print_string (Ast.string_of_program ast)
  | _ -> let sast = Semant.check ast in
    match !action with
      Ast     -> ()
    | Sast   -> print_string (Sast.string_of_sprogram sast)
    | LLVM_IR -> print_string (Llvm.string_of_llmodule
        (Codegen.translate sast))
    | Compile -> let m = Codegen.translate sast in
   Llvm_analysis.assert_valid_module m;
   print_string (Llvm.string_of_llmodule m)
```

```
**************************************************
builtins.c
**************************************************

/*
 *  bugsy's BACKBONE
 */

#include <stdio.h>
#include <GL/glut.h>
#include <math.h>
```

```
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#define pi 3.142857

float s = 1;
int count = 0;
int animationCount = 0;
float l = 0.5;

struct Shape {
    char shape[20];
    char shapeId[10];
    double x;
    double y;
    double n;
    double r;
    double w;
    double h;
    double b;
    double s;
    double x1;
    double y1;
    double x2;
    double y2;
    char stroke[20];
    double thiccness;
    char fill[20];
} Shape;

struct Animation {
    struct Shape shape;
    char animation[20];
    double translateX;
    double translateY;
    double speed;
    double angle;
    double scale;
} Animation;

struct Shape* shapes;
struct Animation* animations;
int id_len = 7;

void printShapes() {
    for(int i = 0; i < count; i++) {
        struct Shape s = shapes[i];
        printf("Shape: %s ID: %s\nx: %d y: %d\nn: %d r: %d\n", s.shape,
            s.shapeId, (int)s.x, (int)s.y, (int)s.n, (int)s.r);
```

```
        printf("w: %d h: %d\nb: %d s: %d\nx1: %d y1: %d\nx2: %d y2:
            %d\n", (int)s.w, (int)s.h, (int)s.b, (int)s.s, (int)s.x1,
            (int)s.y1, (int)s.x2, (int)s.y2);
        printf("stroke: %s thickness: %d\nfill: %s\n", s.stroke,
            (int)s.thiccness, s.fill);
    }
}

void myInit () {
    // making background color black as first
    // 3 arguments all are 0.0
    glClearColor(0.0, 0.0, 0.0, 1.0);

    // making picture color green (in RGB mode), as middle argument is
        1.0
    glColor3f(0.0, 1.0, 0.0);

    // breadth of picture boundary is 1 pixel
    glPointSize(1.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    // setting window dimension in X- and Y- direction
    gluOrtho2D(-780, 780, -420, 420);
}

void add_point_xy(double x, double y) {
    printf("x: %.2f, y: %.2f\n", x, y);
}

void disp() {
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POINTS);
    float x, y, i;
    for (i = 0; i < (2 * pi); i += 0.001) {
        x = 50 * cos(i) + 10;
        y = 50 * sin(i) + 20;
        glVertex2i(x, y);
    }
    glEnd();
    glFlush();
}

double* str_to_arr(char* str_in) {

    char str[strlen(str_in)];
    sprintf(str, "%s", str_in);

    char* delim = " ";
    char *ptr = strtok(str, delim);
```

```
    double* arr = malloc(3 * sizeof(double));
    int i = 0;
    char* end;

   while(ptr != NULL) {
       arr[i] = strtod(ptr, &end);
     ptr = strtok(NULL, delim);
       i++;
   }

    return arr;

}

void genId(char *dest, size_t length) {
    // generates random IDs for the shapes upon creation
    char charset[] = "0123456789"
                     "abcdefghijklmnopqrstuvwxyz"
                     "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

    while (length-- > 0) {
        size_t index = (double) rand() / RAND_MAX * (sizeof charset - 1);
        *dest++ = charset[index];
    }
    *dest = '\0';
}

char* add_ellipse(double x, double y, double w, double h, char* stroke,
     double thiccness, char* fill, char* id) {

    struct Shape shape;

    char* shapeId = malloc(sizeof(char) * 100);
    if(strcmp(id, "") == 0) {
        size_t len = id_len;
        genId(shapeId, len);
        strcpy(shape.shape, "ellipse");
        strcpy(shape.shapeId, shapeId);
        shape.x = x;
        shape.y = y;
        shape.w = w;
        shape.h = h;

        shape.n = 0;
        shape.r = 0;
        shape.b = 0;
        shape.s = 0;
        shape.x1 = 0;
        shape.y1 = 0;
        shape.x2 = 0;
```

```
    shape.y2 = 0;

    strcpy(shape.stroke, stroke);
    shape.thiccness = thiccness;
    strcpy(shape.fill, fill);

    shapes[count] = shape;
    count++;

} else {
    for(int i = 0; i < count; i++) {
        if(strcmp(shapes[i].shapeId, id) == 0) {
            strcpy(shapes[i].shape, "ellipse");
            strcpy(shapes[i].shapeId, id);
            shapes[i].x = x;
            shapes[i].y = y;
            shapes[i].w = w;
            shapes[i].h = h;
            strcpy(shapes[i].stroke, stroke);
            shapes[i].thiccness = thiccness;
            strcpy(shapes[i].fill, fill);

            for(int j = 0; j < animationCount; j++) {
                if(strcmp(animations[j].shape.shapeId, id) == 0) {
                    animations[j].shape = shapes[i];
                }
            }
        }
    }
}

double* stroke_arr = str_to_arr(stroke);
double* fill_arr = str_to_arr(fill);
int num_segments = 100;
float xi = 1;
float yi = 0;

float theta = 2 * 3.1415926 / num_segments;
float c = cosf(theta); //precalculate the sine and cosine
float s = sinf(theta);

float t;
if(fill_arr[0] >= 0.0) {

    glColor3f(fill_arr[0], fill_arr[1], fill_arr[2]);
    glBegin(GL_TRIANGLE_FAN);
    for(int i = 0; i < num_segments; i++) {
          //apply radius and offset
          glVertex2f(xi * w + x, yi * h + y); //output vertex
```

```
            //apply the rotation matrix
            t = xi;
            xi = c * xi - s * yi;
            yi = s * t + c * yi;
        }
        glEnd();
    }

    if(stroke_arr[0] >= 0.0) {

        glColor3f(stroke_arr[0], stroke_arr[1], stroke_arr[2]);
        glBegin(GL_LINE_LOOP);
        for(int i = 0; i < num_segments; i++) {
            //apply radius and offset
            glVertex2f(xi * w + x, yi * h + y); //output vertex

            //apply the rotation matrix
            t = xi;
            xi = c * xi - s * yi;
            yi = s * t + c * yi;
        }
        glEnd();
    }
    return shapeId;
}

char* add_circle(double x, double y, double r, char* stroke, double
    thiccness, char* fill, char* id) {

    double* stroke_arr = str_to_arr(stroke);
    double* fill_arr = str_to_arr(fill);

    struct Shape shape;
    char* shapeId = malloc(sizeof(char) * 100);
    if(strcmp(id, "") == 0) {
        size_t len = id_len;
        genId(shapeId, len);
        strcpy(shape.shape, "circle");
        strcpy(shape.shapeId, shapeId);
        shape.x = x;
        shape.y = y;
        shape.r = r;

        shape.w = 0;
        shape.h = 0;
        shape.n = 0;
        shape.b = 0;
        shape.s = 0;
        shape.x1 = 0;
        shape.y1 = 0;
```

```
    shape.x2 = 0;
    shape.y2 = 0;

    strcpy(shape.stroke, stroke);
    shape.thiccness = thiccness;
    strcpy(shape.fill, fill);
    shapes[count] = shape;
    count++;

} else {

    for(int i = 0; i < count; i++) {
        if(strcmp(shapes[i].shapeId, id) == 0) {
            strcpy(shapes[i].shape, "circle");
            strcpy(shapes[i].shapeId, id);
            shapes[i].x = x;
            shapes[i].y = y;
            shapes[i].r = r;
            strcpy(shapes[i].stroke, stroke);
            shapes[i].thiccness = thiccness;
            strcpy(shapes[i].fill, fill);

            for(int j = 0; j < animationCount; j++) {
                if(strcmp(animations[j].shape.shapeId, id) == 0) {
                    animations[j].shape = shapes[i];
                }
            }
        }


    }
}

if(fill_arr[0] >= 0.0) {

    glColor3f(fill_arr[0], fill_arr[1], fill_arr[2]);
    glBegin(GL_TRIANGLE_FAN);
    for (float i = 0; i < (2 * pi); i += 0.001) {
        float a = r * cos(i) + x;
        float b = r * sin(i) + y;
        glVertex2i(a, b);
    }
    glEnd();
}

if(stroke_arr[0] >= 0.0) {

    glColor3f(stroke_arr[0], stroke_arr[1], stroke_arr[2]);
    glLineWidth(thiccness);
    glBegin(GL_LINE_LOOP);
```

```
        for (float i = 0; i < (2 * pi); i += 0.1) {
            float a = r * cos(i) + x;
            float b = r * sin(i) + y;
            glVertex2i(a, b);
        }
        glEnd();
    }
    return shapeId;
}

char* add_square(double x, double y, double size, char* stroke, double
     thiccness, char* fill, char* id) {

    double* stroke_arr = str_to_arr(stroke);
    double* fill_arr = str_to_arr(fill);

    struct Shape shape;

    char* shapeId = malloc(sizeof(char) * 100);
    if(strcmp(id, "") == 0) {
        size_t len = id_len;
        genId(shapeId, len);
        strcpy(shape.shape, "square");
        strcpy(shape.shapeId, shapeId);
        shape.x = x;
        shape.y = y;
        shape.s = size;

        shape.w = 0;
        shape.h = 0;
        shape.n = 0;
        shape.b = 0;
        shape.x1 = 0;
        shape.y1 = 0;
        shape.x2 = 0;
        shape.y2 = 0;

        strcpy(shape.stroke, stroke);
        shape.thiccness = thiccness;
        strcpy(shape.fill, fill);

        shapes[count] = shape;
        count++;

    } else {

        for(int i = 0; i < count; i++) {
            struct Shape s = shapes[i];

            if(strcmp(shapes[i].shapeId, id) == 0) {
```

```
            strcpy(shapes[i].shape, "square");
            strcpy(shapes[i].shapeId, id);
            shapes[i].x = x;
            shapes[i].y = y;
            shapes[i].s = size;
            strcpy(shapes[i].stroke, stroke);
            shapes[i].thiccness = thiccness;
            strcpy(shapes[i].fill, fill);

            for(int j = 0; j < animationCount; j++) {
                if(strcmp(animations[j].shape.shapeId, id) == 0) {
                    animations[j].shape = shapes[i];
                }
            }
        }
    }
}

if(fill_arr[0] >= 0.0) {

    glColor3f(fill_arr[0], fill_arr[1], fill_arr[2]);
    glBegin(GL_QUADS);
        glVertex2f(x-(size/2.0), y-(size/2.0));
     glVertex2f(x-(size/2.0), y+(size/2.0));
        glVertex2f(x+(size/2.0), y+(size/2.0));
        glVertex2f(x+(size/2.0), y-(size/2.0));
    glEnd();
}

if(stroke[0] >= 0.0) {

    glColor3f(stroke_arr[0], stroke_arr[1], stroke_arr[2]);
    glLineWidth(thiccness);
    glBegin(GL_LINE_LOOP);
        glVertex2f(x-(size/2.0), y-(size/2.0));
     glVertex2f(x-(size/2.0), y+(size/2.0));
        glVertex2f(x+(size/2.0), y+(size/2.0));
        glVertex2f(x+(size/2.0), y-(size/2.0));
    glEnd();
}
return shapeId;
}

char* add_triangle(double x, double y, double b, double h, char* stroke,
    double thiccness, char* fill, char* id) {

double* stroke_arr = str_to_arr(stroke);
double* fill_arr = str_to_arr(fill);

struct Shape shape;
```

```
char* shapeId = malloc(sizeof(char) * 100);
if(strcmp(id, "") == 0) {
    size_t len = id_len;
    genId(shapeId, len);
    strcpy(shape.shape, "triangle");
    strcpy(shape.shapeId, shapeId);
    shape.x = x;
    shape.y = y;
    shape.b = b;
    shape.h = h;

    shape.w = 0;
    shape.n = 0;
    shape.s = 0;
    shape.x1 = 0;
    shape.y1 = 0;
    shape.x2 = 0;
    shape.y2 = 0;

    strcpy(shape.stroke, stroke);
    shape.thiccness = thiccness;
    strcpy(shape.fill, fill);

    shapes[count] = shape;
    count++;

} else {

    for(int i = 0; i < count; i++) {
        struct Shape s = shapes[i];

        if(strcmp(shapes[i].shapeId, id) == 0) {
            strcpy(shapes[i].shape, "triangle");
            strcpy(shapes[i].shapeId, id);
            shapes[i].x = x;
            shapes[i].y = y;
            shapes[i].b = b;
            shapes[i].h = h;
            strcpy(shapes[i].stroke, stroke);
            shapes[i].thiccness = thiccness;
            strcpy(shapes[i].fill, fill);

            for(int j = 0; j < animationCount; j++) {
                if(strcmp(animations[j].shape.shapeId, id) == 0) {
                    animations[j].shape = shapes[i];
                }
            }
        }
    }
```

```
    }

    if(fill_arr[0] >= 0.0) {

        glColor3f(fill_arr[0], fill_arr[1], fill_arr[2]);
        glBegin(GL_TRIANGLES);
            glVertex2f(x, y+(h/2.0));
         glVertex2f(x-(b/2.0), y-(h/2.0));
            glVertex2f(x+(b/2.0), y-(h/2.0));
        glEnd();

    }

    if(stroke[0] >= 0.0) {

        glColor3f(stroke_arr[0], stroke_arr[1], stroke_arr[2]);
        glLineWidth(thiccness);
        glBegin(GL_LINE_LOOP);
            glVertex2f(x, y+(h/2.0));
            glVertex2f(x-(b/2.0), y-(h/2.0));
            glVertex2f(x+(b/2.0), y-(h/2.0));
        glEnd();

    }
    return shapeId;
}

char* add_rectangle(double x, double y, double w, double h, char*
     stroke, double thiccness, char* fill, char* id) {

    double* stroke_arr = str_to_arr(stroke);
    double* fill_arr = str_to_arr(fill);

    struct Shape shape;

    char* shapeId = malloc(sizeof(char) * 100);
    if(strcmp(id, "") == 0) {
        size_t len = id_len;
        genId(shapeId, len);
        strcpy(shape.shape, "rectangle");
        strcpy(shape.shapeId, shapeId);
        shape.x = x;
        shape.y = y;
        shape.w = w;
        shape.h = h;

        shape.n = 0;
        shape.b = 0;
        shape.s = 0;
        shape.x1 = 0;
```

```
    shape.y1 = 0;
    shape.x2 = 0;
    shape.y2 = 0;

    strcpy(shape.stroke, stroke);
    shape.thiccness = thiccness;
    strcpy(shape.fill, fill);

    shapes[count] = shape;
    count++;

} else {

    for(int i = 0; i < count; i++) {
        struct Shape s = shapes[i];

        if(strcmp(shapes[i].shapeId, id) == 0) {
            strcpy(shapes[i].shape, "rectangle");
            strcpy(shapes[i].shapeId, id);
            shapes[i].x = x;
            shapes[i].y = y;
            shapes[i].w = w;
            shapes[i].h = h;
            strcpy(shapes[i].stroke, stroke);
            shapes[i].thiccness = thiccness;
            strcpy(shapes[i].fill, fill);

            for(int j = 0; j < animationCount; j++) {
                if(strcmp(animations[j].shape.shapeId, id) == 0) {
                    animations[j].shape = shapes[i];
                }
            }
        }
    }
}

if(fill_arr[0] >= 0.0) {

    glColor3f(fill_arr[0], fill_arr[1], fill_arr[2]);
    glBegin(GL_QUADS);
        glVertex2f(x-(w/2.0), y-(h/2.0));
     glVertex2f(x-(w/2.0), y+(h/2.0));
        glVertex2f(x+(w/2.0), y+(h/2.0));
        glVertex2f(x+(w/2.0), y-(h/2.0));
    glEnd();
}

if(stroke_arr[0] >= 0.0) {
    glColor3f(stroke_arr[0], stroke_arr[1], stroke_arr[2]);
    glLineWidth(thiccness);
```

```
        glBegin(GL_LINE_LOOP);
            glVertex2f(x-(w/2.0), y-(h/2.0));
            glVertex2f(x-(w/2.0), y+(h/2.0));
            glVertex2f(x+(w/2.0), y+(h/2.0));
            glVertex2f(x+(w/2.0), y-(h/2.0));
        glEnd();
    }
    return shapeId;
}

char* add_line(double x1, double y1, double x2, double y2, char* stroke,
     double thiccness, char* id) {

    double* stroke_arr = str_to_arr(stroke);

    struct Shape shape;

    char* shapeId = malloc(sizeof(char) * 100);
    if(strcmp(id, "") == 0) {
        size_t len = 100;
        genId(shapeId, len);
        strcpy(shape.shape, "line");
        strcpy(shape.shapeId, shapeId);
        shape.x1 = x1;
        shape.y1 = y1;
        shape.x2 = x2;
        shape.y2 = y2;
        strcpy(shape.stroke, stroke);
        shape.thiccness = thiccness;

        shapes[count] = shape;
        count++;

    } else {

        for(int i = 0; i < count; i++) {
            struct Shape s = shapes[i];

            if(strcmp(shapes[i].shapeId, id) == 0) {
                strcpy(shapes[i].shape, "line");
                strcpy(shapes[i].shapeId, id);
                shapes[i].x1 = x1;
                shapes[i].y1 = y1;
                shapes[i].x2 = x2;
                shapes[i].y2 = y2;
                strcpy(shapes[i].stroke, stroke);
                shapes[i].thiccness = thiccness;

                for(int j = 0; j < animationCount; j++) {
                    if(strcmp(animations[j].shape.shapeId, id) == 0) {
```

```
                    animations[j].shape = shapes[i];
                }
            }
        }
    }

    if(stroke_arr[0] >= 0.0) {
        glColor3f(stroke_arr[0], stroke_arr[1], stroke_arr[2]);
        glLineWidth(thiccness);
        glBegin(GL_LINES);
            glVertex2f(x1, y1);
            glVertex2f(x2, y2);
        glEnd();
    }
    return shapeId;
}

char* add_regagon(double x, double y, double n_in, double r, char*
     stroke, double thiccness, char* fill, char* id) {

    double* stroke_arr = str_to_arr(stroke);
    double* fill_arr = str_to_arr(fill);
    int n = (int)n_in;
    double a = 2 / (pi * n);

    struct Shape shape;
    char* shapeId = malloc(sizeof(char) * id_len);
    if(strcmp(id, "") == 0) {
        size_t len = id_len;
        genId(shapeId, len);
        strcpy(shape.shape, "regagon");
        strcpy(shape.shapeId, shapeId);
        shape.x = x;
        shape.y = y;
        shape.n = n;
        shape.r = r;
        strcpy(shape.stroke, stroke);
        shape.thiccness = thiccness;
        strcpy(shape.fill, fill);

        shapes[count] = shape;
        count++;

    } else {

        for(int i = 0; i < count; i++) {
            struct Shape s = *shapes;

            if(strcmp(shapes[i].shapeId, id) == 0) {
```

```
                strcpy(shapes[i].shape, "regagon");
                strcpy(shapes[i].shapeId, id);
                shapes[i].x = x;
                shapes[i].y = y;
                shapes[i].n = n;
                shapes[i].r = r;
                strcpy(shapes[i].stroke, stroke);
                shapes[i].thiccness = thiccness;
                strcpy(shapes[i].fill, fill);

                for(int j = 0; j < animationCount; j++) {
                    if(strcmp(animations[j].shape.shapeId, id) == 0) {
                        animations[j].shape = shapes[i];
                    }
                }
            }
        }
    }

    if(fill_arr[0] >= 0.0) {

        glColor3f(fill_arr[0], fill_arr[1], fill_arr[2]);
        glBegin(GL_POLYGON);
            for(int i = 0; i < n; i++) {
                double x_i = x + r * cos(2 * pi * i / n);
                double y_i = y + r * sin(2 * pi * i / n);
                glVertex2f(x_i, y_i);
            }
        glEnd();
    }

    if(stroke_arr[0] >= 0.0) {
        glColor3f(stroke_arr[0], stroke_arr[1], stroke_arr[2]);
        glLineWidth(thiccness);
        glBegin(GL_LINE_LOOP);
            for(int i = 0; i < n; i++) {
                double x_i = x + r * cos(2 * pi * i / n);
                double y_i = y + r * sin(2 * pi * i / n);
                glVertex2f(x_i, y_i);
            }
        glEnd();
    }
    return shapeId;

}

void scaleBy(struct Shape shape, double scale, double speed) {

    double scaled = 0.0;
    double inc = 0.01;
```

```
double abs_val = scale - 1;
if(abs_val < 0) {
    abs_val *= -1;
}
double time = (speed * 1000000) / (abs_val / inc);

if(scale < 1) {
    scaled = 1.0;
}

double wi = 0.0;
double hi = 0.0;
double si = 0.0;
double ri = 0.0;
double bi = 0.0;

if(strcmp(shape.shape, "ellipse") == 0) {
    wi = shape.w;
    hi = shape.h;
} else if(strcmp(shape.shape, "circle") == 0) {
    ri = shape.r;
} else if(strcmp(shape.shape, "square") == 0) {
    si = shape.s;
} else if(strcmp(shape.shape, "triangle") == 0) {
    bi = shape.b;
    hi = shape.h;
} else if(strcmp(shape.shape, "rectangle") == 0) {
    wi = shape.w;
    hi = shape.h;
} else if(strcmp(shape.shape, "regagon") == 0) {
    ri = shape.r;
}

while(1) {
    if(scale >= 1) {
        if(scaled >= scale - 1) {
            break;
        }
    } else {
        if(scaled <= scale) {
            break;
        }
    }
    glClear(GL_COLOR_BUFFER_BIT);

    for(int i = 0; i < count; i++) {
        struct Shape s = shapes[i];
        if(strcmp(shapes[i].shapeId, shape.shapeId) != 0) {
            if(strcmp(shapes[i].shape, "ellipse") == 0) {
                add_ellipse(shapes[i].x,
```

```
                    shapes[i].y, shapes[i].w, shapes[i].h,
                    shapes[i].stroke,
                    shapes[i].thiccness, shapes[i].fill,
                        shapes[i].shapeId);
        } else if(strcmp(shapes[i].shape, "circle") == 0) {
            add_circle(shapes[i].x,
                    shapes[i].y, shapes[i].r,
                    shapes[i].stroke,
                    shapes[i].thiccness, shapes[i].fill,
                        shapes[i].shapeId);
        } else if(strcmp(shapes[i].shape, "square") == 0) {
            add_square(shapes[i].x,
                    shapes[i].y, shapes[i].s,
                    shapes[i].stroke,
                    shapes[i].thiccness, shapes[i].fill,
                        shapes[i].shapeId);
        } else if(strcmp(shapes[i].shape, "triangle") == 0) {
            add_triangle(shapes[i].x,
                    shapes[i].y, shapes[i].b, shapes[i].h,
                    shapes[i].stroke,
                    shapes[i].thiccness, shapes[i].fill,
                        shapes[i].shapeId);
        } else if(strcmp(shapes[i].shape, "rectangle") == 0) {
            add_rectangle(shapes[i].x,
                    shapes[i].y, shapes[i].w, shapes[i].h,
                    shapes[i].stroke,
                    shapes[i].thiccness, shapes[i].fill,
                        shapes[i].shapeId);
        } else if(strcmp(shapes[i].shape, "regagon") == 0) {
            add_regagon(shapes[i].x,
                    shapes[i].y, shapes[i].n, shapes[i].r,
                    shapes[i].stroke,
                    shapes[i].thiccness, shapes[i].fill,
                        shapes[i].shapeId);
        }
    }
}

if(scale >= 1) {
    scaled += inc;
} else {
    scaled -= inc;
}

if(strcmp(shape.shape, "ellipse") == 0) {
    if(scale >= 1) {
        add_ellipse(shape.x, shape.y, wi * (1 + scaled), hi * (1
            + scaled),
               shape.stroke, shape.thiccness, shape.fill,
                   shape.shapeId);
```

```
    } else {
        add_ellipse(shape.x, shape.y, wi * scaled, hi * scaled,
               shape.stroke, shape.thiccness, shape.fill,
                   shape.shapeId);
    }
} else if(strcmp(shape.shape, "circle") == 0) {
    if(scale >= 1) {
        add_circle(shape.x, shape.y, ri * (1 + scaled),
               shape.stroke, shape.thiccness, shape.fill,
                   shape.shapeId);
    } else {
        add_circle(shape.x, shape.y, ri * scaled,
               shape.stroke, shape.thiccness, shape.fill,
                   shape.shapeId);
    }
} else if(strcmp(shape.shape, "square") == 0) {
    if(scale >= 1) {
        add_square(shape.x, shape.y, si * (1 + scaled),
               shape.stroke, shape.thiccness, shape.fill,
                   shape.shapeId);
    } else {
        add_square(shape.x, shape.y, si * scaled,
               shape.stroke, shape.thiccness, shape.fill,
                   shape.shapeId);
    }
} else if(strcmp(shape.shape, "triangle") == 0) {
    if(scale >= 1) {
        add_triangle(shape.x, shape.y, bi * (1 + scaled), hi * (1
            + scaled),
               shape.stroke, shape.thiccness, shape.fill,
                   shape.shapeId);
    } else {
        add_triangle(shape.x, shape.y, bi * scaled, hi * scaled,
               shape.stroke, shape.thiccness, shape.fill,
                   shape.shapeId);
    }
} else if(strcmp(shape.shape, "rectangle") == 0) {
    if(scale >= 1) {
        add_rectangle(shape.x, shape.y, wi * (1 + scaled), hi *
            (1 + scaled),
               shape.stroke, shape.thiccness, shape.fill,
                   shape.shapeId);
    } else {
        add_rectangle(shape.x, shape.y, wi * scaled, hi * scaled,
               shape.stroke, shape.thiccness, shape.fill,
                   shape.shapeId);
    }
} else if(strcmp(shape.shape, "regagon") == 0) {
    if(scale >= 1) {
        add_regagon(shape.x, shape.y, shape.n, ri * (1 + scaled),
```

```
                    shape.stroke, shape.thiccness, shape.fill,
                        shape.shapeId);
            } else {
                add_regagon(shape.x, shape.y, shape.n, ri * scaled,
                    shape.stroke, shape.thiccness, shape.fill,
                        shape.shapeId);
            }
        }

        glFlush();
        usleep(time);
    }

}

void scaleById(char* id, double scale, double speed) {
    for(int i = 0; i < count; i++) {
        if(strcmp(shapes[i].shapeId, id) == 0) {
            struct Animation a;
            a.shape = shapes[i];
            strcpy(a.animation, "scale");
            a.scale = scale;
            a.speed = speed;
            animations[animationCount] = a;
            animationCount++;
        }
    }
}

void rotateBy(struct Shape shape, double angle, double speed) {

    double rotated = 0.0;
    double inc = 0.1 * (angle / abs(angle));

    double abs_val = angle;
    if(abs_val < 0) {
        abs_val *= -1;
    }
    int time = (int) ((speed * 1000000) / (abs_val / abs(inc)));
    double x_orig = shape.x;
    double y_orig = shape.y;

    struct Shape* front = shapes;
    while(rotated < angle) {
        glClear(GL_COLOR_BUFFER_BIT);
        shapes = front;
        for(int i = 0; i < count; i++) {
            struct Shape s = shapes[i];

            if(strcmp(s.shapeId, shape.shapeId) != 0) {
```

```
        if(strcmp(s.shape, "ellipse") == 0) {
            add_ellipse(s.x,
                        s.y, s.w, s.h,
                        s.stroke,
                        s.thiccness, s.fill, s.shapeId);
        } else if(strcmp(s.shape, "circle") == 0) {
            add_circle(s.x,
                        s.y, s.r,
                        s.stroke,
                        s.thiccness, s.fill, s.shapeId);
        } else if(strcmp(s.shape, "square") == 0) {
            add_square(s.x,
                        s.y, s.s,
                        s.stroke,
                        s.thiccness, s.fill, s.shapeId);
        } else if(strcmp(s.shape, "triangle") == 0) {
            add_triangle(s.x,
                        s.y, s.b, s.h,
                        s.stroke,
                        s.thiccness, s.fill, s.shapeId);
        } else if(strcmp(s.shape, "rectangle") == 0) {
            add_rectangle(s.x,
                        s.y, s.w, s.h,
                        s.stroke,
                        s.thiccness, s.fill, s.shapeId);
        } else if(strcmp(s.shape, "regagon") == 0) {
            add_regagon(s.x,
                        s.y, s.n, s.r,
                        s.stroke,
                        s.thiccness, s.fill, s.shapeId);
        }
    }
}

rotated += inc;

glPushMatrix();
glTranslatef(shape.x, shape.y, 0);
glRotatef(rotated, 0, 0, 1);

if(strcmp(shape.shape, "ellipse") == 0) {
    add_ellipse(0, 0, shape.w, shape.h, shape.stroke,
        shape.thiccness, shape.fill, shape.shapeId);
} else if(strcmp(shape.shape, "circle") == 0) {
    add_circle(0, 0, shape.r, shape.stroke, shape.thiccness,
        shape.fill, shape.shapeId);
} else if(strcmp(shape.shape, "square") == 0) {
    add_square(0, 0, shape.s, shape.stroke, shape.thiccness,
        shape.fill, shape.shapeId);
} else if(strcmp(shape.shape, "triangle") == 0) {
```

```
            add_triangle(0, 0, shape.b, shape.h, shape.stroke,
                shape.thiccness, shape.fill, shape.shapeId);
        } else if(strcmp(shape.shape, "rectangle") == 0) {
            add_rectangle(0, 0, shape.w, shape.h, shape.stroke,
                shape.thiccness, shape.fill, shape.shapeId);
        } else if(strcmp(shape.shape, "regagon") == 0) {
            add_regagon(0, 0, shape.n, shape.r, shape.stroke,
                shape.thiccness, shape.fill, shape.shapeId);
        }

        glPopMatrix();

        glFlush();
        usleep(time);
    }
}

void rotateById(char* id, double angle, double speed) {
    for(int i = 0; i < count; i++) {
        if(strcmp(shapes[i].shapeId, id) == 0) {
            struct Animation a;
            a.shape = shapes[i];
            strcpy(a.animation, "rotate");
            a.angle = angle;
            a.speed = speed;
            animations[animationCount] = a;
            animationCount++;
        }
    }
}

void moveBy(struct Shape shape, double translateX, double translateY,
     double speed) {
    double movedX = 0.0;
    double movedY = 0.0;
    double incX = 1 * (translateX / abs(translateX));
    double incY = 1 * (translateY / abs(translateY));

    if(translateX > translateY) {
        incX = translateX / abs(translateY);
    } else {
        incY = translateY / abs(translateX);
    }

    double abs_val = translateX;
    if(abs_val < 0) {
        abs_val *= -1;
    }

    double time = (speed * 1000000) / (abs_val / abs(incX));
```

```
struct Shape* front = shapes;
while(1) {
    if(translateX < 0) {
        if(movedX <= translateX) {
            break;
        }
    } else {
        if(movedX >= translateX) {
            break;
        }
    }
    glClear(GL_COLOR_BUFFER_BIT);
    shapes = front;
    for(int i = 0; i < count; i++) {
        struct Shape s = shapes[i];

        if(strcmp(s.shapeId, shape.shapeId) != 0) {
            if(strcmp(s.shape, "ellipse") == 0) {
                add_ellipse(s.x,
                            s.y, s.w, s.h,
                            s.stroke,
                            s.thiccness, s.fill, s.shapeId);
            } else if(strcmp(s.shape, "circle") == 0) {
                add_circle(s.x,
                            s.y, s.r,
                            s.stroke,
                            s.thiccness, s.fill, s.shapeId);
            } else if(strcmp(s.shape, "square") == 0) {
                add_square(s.x,
                            s.y, s.s,
                            s.stroke,
                            s.thiccness, s.fill, s.shapeId);
            } else if(strcmp(s.shape, "triangle") == 0) {
                add_triangle(s.x,
                            s.y, s.b, s.h,
                            s.stroke,
                            s.thiccness, s.fill, s.shapeId);
            } else if(strcmp(s.shape, "rectangle") == 0) {
                add_rectangle(s.x,
                            s.y, s.w, s.h,
                            s.stroke,
                            s.thiccness, s.fill, s.shapeId);
            } else if(strcmp(s.shape, "regagon") == 0) {
                add_regagon(s.x,
                            s.y, s.n, s.r,
                            s.stroke,
                            s.thiccness, s.fill, s.shapeId);
            } else if(strcmp(s.shape, "line") == 0) {
                add_line(s.x1,
                            s.y1, s.x2, s.y2,
```

```
                        s.stroke,
                        s.thiccness, s.shapeId);
                }
            }
        }

        movedX += incX;
        movedY += incY;

        if(strcmp(shape.shape, "ellipse") == 0) {
            add_ellipse(shape.x + movedX, shape.y + movedY, shape.w,
                shape.h, shape.stroke, shape.thiccness, shape.fill,
                shape.shapeId);
        } else if(strcmp(shape.shape, "circle") == 0) {
            add_circle(shape.x + movedX, shape.y + movedY, shape.r,
                shape.stroke, shape.thiccness, shape.fill,
                shape.shapeId);
        } else if(strcmp(shape.shape, "square") == 0) {
            add_square(shape.x + movedX, shape.y + movedY, shape.s,
                shape.stroke, shape.thiccness, shape.fill,
                shape.shapeId);
        } else if(strcmp(shape.shape, "triangle") == 0) {
            add_triangle(shape.x + movedX, shape.y + movedY, shape.b,
                shape.h, shape.stroke, shape.thiccness, shape.fill,
                shape.shapeId);
        } else if(strcmp(shape.shape, "rectangle") == 0) {
            add_rectangle(shape.x + movedX, shape.y + movedY, shape.w,
                shape.h, shape.stroke, shape.thiccness, shape.fill,
                shape.shapeId);
        } else if(strcmp(shape.shape, "regagon") == 0) {
            add_regagon(shape.x + movedX, shape.y + movedY, shape.n,
                shape.r, shape.stroke, shape.thiccness, shape.fill,
                shape.shapeId);
        } else if(strcmp(shape.shape, "line") == 0) {
            add_line(shape.x1 + movedX, shape.y1 + movedY, shape.x2 +
                movedX, shape.y2 + movedY, shape.stroke,
                shape.thiccness, shape.shapeId);
        }

        glFlush();
        usleep(time);
    }
}

void moveById(char* id, double translateX, double translateY, double
    speed) {
    for(int i = 0; i < count; i++) {
        if(strcmp(shapes[i].shapeId, id) == 0) {
            struct Animation a;
            a.shape = shapes[i];
```

```
            strcpy(a.animation, "move");
            a.translateX = translateX;
            a.translateY = translateY;
            a.speed = speed;
            animations[animationCount] = a;
            animationCount++;
        }
    }
}

double map(double a, double b, double x) {
    return (x - a) / (b - a) ;
}

void init_canvas() {
    shapes = malloc(sizeof(struct Shape) * 100);
    animations = malloc(sizeof(struct Animation) * 100);
}

void add_canvas(double width, double height, double xOffset, double
     yOffset) {

    int c = 0;
    char ** args;
    double aspect = width / height;
    // shapes = malloc(sizeof(struct Shape) * 100);

    glutInit(&c, args);

    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

    // giving window size in X- and Y- direction
    int w = (int) width;
    int h = (int) height;
    int x = (int) xOffset;
    int y = (int) yOffset;

    glutInitWindowSize(w, h);

    glutInitWindowPosition(x, y);

    glutCreateWindow("Canvas");

    myInit();

    glClearColor(0.0, 0.0, 0.0, 0.0);   // black background
    glMatrixMode(GL_PROJECTION);        // setup viewing projection
    glLoadIdentity();                   // start with identity matrix
    glOrtho(0.0, w, 0.0, h, 0.0, 1.0);  // setup a 10x10x2 viewing world
```

```
printShapes();
for(int i = 0; i < count; i++) {
    struct Shape s = shapes[i];
    if(strcmp(s.shape, "ellipse") == 0) {
        add_ellipse(s.x,
                    s.y, s.w, s.h,
                    s.stroke,
                    s.thiccness, s.fill, s.shapeId);
    } else if(strcmp(s.shape, "circle") == 0) {
        add_circle(s.x,
                    s.y, s.r,
                    s.stroke,
                    s.thiccness, s.fill, s.shapeId);
    } else if(strcmp(s.shape, "square") == 0) {
        add_square(s.x,
                    s.y, s.s,
                    s.stroke,
                    s.thiccness, s.fill, s.shapeId);
    } else if(strcmp(s.shape, "triangle") == 0) {
        add_triangle(s.x,
                    s.y, s.b, s.h,
                    s.stroke,
                    s.thiccness, s.fill, s.shapeId);
    } else if(strcmp(s.shape, "rectangle") == 0) {
        add_rectangle(s.x,
                    s.y, s.w, s.h,
                    s.stroke,
                    s.thiccness, s.fill, s.shapeId);
    } else if(strcmp(s.shape, "regagon") == 0) {
        add_regagon(s.x,
                    s.y, s.n, s.r,
                    s.stroke,
                    s.thiccness, s.fill, s.shapeId);
    } else if(strcmp(s.shape, "line") == 0) {
        add_line(s.x1,
                    s.y1, s.x2, s.y2,
                    s.stroke,
                    s.thiccness, s.shapeId);
    }
}

for(int i = 0; i < animationCount; i++) {

    if(strcmp(animations[i].animation, "move") == 0) {
        moveBy(animations[i].shape, animations[i].translateX,
            animations[i].translateY, animations[i].speed);
    }
    else if(strcmp(animations[i].animation, "rotate") == 0) {
        rotateBy(animations[i].shape, animations[i].angle,
            animations[i].speed);
```

```
        }
        else if(strcmp(animations[i].animation, "scale") == 0) {
            scaleBy(animations[i].shape, animations[i].scale,
                animations[i].speed);
        }
    }

    printShapes();

    glFlush();
    if(strcmp(getenv("DEBUG"), "1") != 0) {
        glutMainLoop();
    }

}

#ifdef BUILD_TEST
int main()
{
  char s[] = "HELLO WORLD09AZ";
  char *c;
  for ( c = s ; *c ; c++) printbig(*c);
}
#endif


**************************************************
codegen.ml
**************************************************

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

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

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

Detailed documentation on the OCaml LLVM library:

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

*)

module L = Llvm
module A = Ast
open Sast


module StringMap = Map.Make(String)
```

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

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

  let convert_int = function
      A.IntLiteral(i) -> i
      | A.NumLit(i) -> int_of_string i
      | _ -> raise (Failure "Something went wrong")
  in

  (* Get types from the context *)
  let i32_t     = L.i32_type   context
  and i8_t      = L.i8_type    context
  and i1_t      = L.i1_type    context
  and float_t   = L.double_type context
  and string_t  = L.pointer_type (L.i8_type context) (*new string type *)
  and array_t   = L.array_type
  and void_t    = L.void_type  context
  and struct_t arr = L.struct_type context arr
  in

  (* Return the LLVM type for a Bugsy type *)
  let rec ltype_of_typ = function
      A.Num  -> float_t
    | A.Bool -> i1_t
    | A.Void -> void_t
    | A.String -> string_t
    | A.Int -> i32_t
    | A.Array(typ, size) -> (match typ with
          A.Num -> array_t float_t (convert_int size)
        | _ -> raise (Failure "Error: Not implemented in codegen")
    )
    | A.Object(classTyp) -> L.pointer_type (struct_t (struct_t_to_arr
        classTyp))

  and struct_t_to_arr classTyp =
    let varTypes = List.map (fun (_, (t,_)) -> t) (StringMap.bindings
        classTyp.A.instanceVars) in
    Array.of_list (List.map ltype_of_typ varTypes)
  in



  let object_type = L.named_struct_type context "obj" in
  let objectptr_type = L.pointer_type object_type in
```

```
let objectref_type = L.named_struct_type context "objref" in
L.struct_set_body objectref_type [|i32_t; objectptr_type|] false;


(*go through all functions, find main, and change main to return int *)
(*let functions = List.map (fun x -> (x.styp <- A.Int); x)
    camFunctions in *)
let functions = List.map (fun x -> if x.sfname = "main" then ((x.styp
    <- A.Int); x) else x) functions' in


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

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

let demo_t : L.lltype =
    L.function_type float_t [||] in
let demo_func : L.llvalue =
    L.declare_function "demo" demo_t the_module in

let add_point_xy_t : L.lltype =
    L.function_type float_t [| float_t; float_t; string_t |] in
let add_point_xy_func : L.llvalue =
    L.declare_function "add_point_xy" add_point_xy_t the_module in

let add_circle_t : L.lltype =
    L.function_type string_t [| float_t; float_t; float_t; string_t;
    float_t; string_t; string_t |] in
let add_circle_func : L.llvalue =
    L.declare_function "add_circle" add_circle_t the_module in

let add_triangle_t : L.lltype =
    L.function_type string_t [| float_t; float_t; float_t; float_t;
        string_t;
    float_t; string_t; string_t |] in
let add_triangle_func : L.llvalue =
    L.declare_function "add_triangle" add_triangle_t the_module in

let add_square_t : L.lltype =
```

```
    L.function_type string_t [| float_t; float_t; float_t; string_t;
        float_t; string_t; string_t |] in
let add_square_func : L.llvalue =
    L.declare_function "add_square" add_square_t the_module in

let add_rectangle_t : L.lltype =
    L.function_type string_t [| float_t; float_t; float_t; float_t;
        string_t;
    float_t; string_t; string_t |] in
let add_rectangle_func : L.llvalue =
    L.declare_function "add_rectangle" add_rectangle_t the_module in

let add_regagon_t : L.lltype =
    L.function_type string_t [| float_t; float_t; float_t; float_t;
        string_t;
    float_t; string_t; string_t |] in
let add_regagon_func : L.llvalue =
    L.declare_function "add_regagon" add_regagon_t the_module in

let add_ellipse_t : L.lltype =
    L.function_type string_t [| float_t; float_t; float_t; float_t;
        string_t;
    float_t; string_t; string_t |] in
let add_ellipse_func : L.llvalue =
    L.declare_function "add_ellipse" add_ellipse_t the_module in

let canvas_t : L.lltype =
    L.function_type float_t [| float_t; float_t; float_t; float_t |] in
let canvas_func : L.llvalue =
    L.declare_function "add_canvas" canvas_t the_module in

let moveById_t : L.lltype =
    L.function_type float_t [| string_t; float_t; float_t; float_t |]
        in
let moveById_func : L.llvalue =
    L.declare_function "moveById" moveById_t the_module in

let scaleById_t : L.lltype =
  L.function_type float_t [| string_t; float_t; float_t |] in
let scaleById_func : L.llvalue =
    L.declare_function "scaleById" scaleById_t the_module in

let rotateById_t : L.lltype =
  L.function_type float_t [| string_t; float_t; float_t |] in
let rotateById_func : L.llvalue =
    L.declare_function "rotateById" rotateById_t the_module in

let init_canvas_t : L.lltype =
    L.function_type float_t [| |] in
let init_canvas_func : L.llvalue =
```

```
    L.declare_function "init_canvas" init_canvas_t the_module in

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

    let ret_type = ftype in

    StringMap.add name (L.define_function name ret_type the_module,
        fdecl) m in

       List.fold_left function_decl StringMap.empty functions in


let find_func s =

       try begin StringMap.find s function_decls; end

  with Not_found -> raise (Failure ("unrecognized function " ^ s))
in

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

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

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

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

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

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


 (* Construct code for an expression; return its value *)
 let rec expr builder ((_, e) : sexpr) = match e with
     SIntLiteral i -> L.const_int i32_t i
   | SStrLit s -> L.build_global_stringptr s "str" builder
   | SBoolLit b -> L.const_int i1_t (if b then 1 else 0)
   | SNumLit nl -> L.const_float_of_string float_t nl
   | SNoexpr    -> L.const_int i32_t 0

   | SArrayAccess(a, e, _) -> let valu = (expr builder e) in

 (*  convert nums to ints *)
 let truncated = L.build_fptosi (valu) i32_t "trunc" builder in

 (* get element pointer to element we're accessing *)
 let result = L.build_in_bounds_gep (lookup a) [| L.const_int i32_t
     0; truncated |] a builder in L.build_load result a builder


   | SArrayAssign (s, e1, e2) ->
     (* let left_value = (expr builder e1) in

      let left_trunc = L.build_fptosi (left_value) i32_t "trunc"
          builder in


      let left_real = L.build_gep (lookup s) [| L.const_int i32_t 0;
          L.const_int i32_t left_trunc |] s builder in
```

```
   let right_value = (expr builder e2) in

   let right_truncated = L.build_fptosi (right_value) i32_t
       "trunc" builder in

       ignore (L.build_store right_value left_trunc builder);
           right_truncated *)

   let left_value = (expr builder e1) in

   let index = L.build_fptosi (left_value) i32_t "trunc" builder in


   let left_real = L.build_gep (lookup s) [| L.const_int i32_t 0;
       index |] s builder in


   let right_value = (expr builder e2) in

   let right_truncated = L.build_fptosi (right_value) i32_t
       "trunc" builder in

       ignore (L.build_store right_value left_real builder);
           right_truncated
| SId s      -> L.build_load (lookup s) s builder
| SArrayLiteral (l, t) -> L.const_array (ltype_of_typ t)
    (Array.of_list (List.map (expr builder) l))
| SAssign (s, e) -> let e' = expr builder e in
                 ignore(L.build_store e' (lookup s) builder); e'
| SConstruct(_,_) -> raise (Failure ("SConstruct not ready yet"))
| SClassCall(_, _, _) -> raise (Failure ("SClassCall not ready
    yet"))
| SAccess(_, _) -> raise (Failure ("SAccess not ready yet"))
| SCrementop(e, op) ->
   let e' = expr builder e in
   let one = expr builder (A.Num, (Sast.SNumLit "1.0")) in
   let eplus = L.build_fadd e' one "tmp" builder
   and eminus = L.build_fsub e' one "tmp" builder
   and s = (match snd e with
     SId s -> s
     | _ -> raise (Failure ("assignment failed")))
   in
   (match op with
     A.PreInc -> ignore(L.build_store eplus (lookup s) builder);
         eplus
   | A.PostInc -> ignore(L.build_store eplus (lookup s) builder);
       e'
   | A.PreDec -> ignore(L.build_store eminus (lookup s) builder);
       eminus
```

```
      | A.PostDec -> ignore(L.build_store eminus (lookup s) builder);
          e'
      | _ -> raise (Failure "Error: Not implemented in codegen")
      )

  | SBinop ((A.Num,_ ) as e1, op, e2) ->
 let e1' = expr builder e1
 and e2' = expr builder e2 in
 (match op with
   A.Add    -> L.build_fadd
 | A.Sub    -> L.build_fsub
 | A.Mult   -> L.build_fmul
 | A.Div    -> L.build_fdiv
 | A.Equal  -> L.build_fcmp L.Fcmp.Oeq
 | A.Neq    -> L.build_fcmp L.Fcmp.One
 | A.Less   -> L.build_fcmp L.Fcmp.Olt
 | A.Leq    -> L.build_fcmp L.Fcmp.Ole
 | A.Greater -> L.build_fcmp L.Fcmp.Ogt
 | A.Geq    -> L.build_fcmp L.Fcmp.Oge
 | A.And | A.Or ->
     raise (Failure "internal error: semant should have rejected
         and/or on float")
| _ -> raise (Failure "Error: Not implemented in codegen")
 ) e1' e2' "tmp" builder
  | SBinop (e1, op, e2) ->
 let e1' = expr builder e1
 and e2' = expr builder e2 in
 (match op with
  A.Add    -> L.build_add
 | A.Sub    -> L.build_sub
 | A.Mult   -> L.build_mul
| A.Div    -> L.build_sdiv
 | A.And    -> L.build_and
 | A.Or     -> L.build_or
 | A.Equal  -> L.build_icmp L.Icmp.Eq
 | A.Neq    -> L.build_icmp L.Icmp.Ne
 | A.Less   -> L.build_icmp L.Icmp.Slt
 | A.Leq    -> L.build_icmp L.Icmp.Sle
 | A.Greater -> L.build_icmp L.Icmp.Sgt
 | A.Geq    -> L.build_icmp L.Icmp.Sge
| _ -> raise (Failure "Error: Not implemented in codegen")
 ) e1' e2' "tmp" builder
  | SUnop(op, ((t, _) as e)) ->
      let e' = expr builder e in
 (match op with
   A.Neg when t = A.Num -> L.build_fneg
 | A.Neg              -> L.build_neg
| A.Not              -> L.build_not) e' "tmp" builder
| SCall ("printb", [e]) ->
  L.build_call printf_func [| int_format_str ; (expr builder e) |]
```

```
  "printf" builder
| SCall ("print", [e]) ->
   L.build_call printf_func [| float_format_str ; (expr builder e) |]
   "printf" builder
| SCall ("demo", []) ->
 L.build_call demo_func [||] "demo" builder
| SCall ("printf", [e]) ->
   L.build_call printf_func [| string_format_str ; (expr builder e)
       |]
   "printf" builder
| SCall ("add_point_xy", [e1; e2; e3]) ->
  L.build_call add_point_xy_func [| (expr builder e1); (expr builder
      e2); (expr builder e3)|]
  "add_point_xy" builder
| SCall ("add_circle", [e1; e2; e3; e4; e5; e6; e7]) ->
  L.build_call add_circle_func [| (expr builder e1); (expr builder
      e2); (expr builder e3);
                            (expr builder e4); (expr builder e5);
                                (expr builder e6);
                            (expr builder e7); |]
"add_circle" builder
| SCall ("add_square", [e1; e2; e3; e4; e5; e6; e7]) ->
  L.build_call add_square_func [| (expr builder e1); (expr builder
      e2); (expr builder e3);
                            (expr builder e4); (expr builder e5);
                                (expr builder e6);
                            (expr builder e7); |]
  "add_square" builder
| SCall ("add_triangle", [e1; e2; e3; e4; e5; e6; e7; e8]) ->
  L.build_call add_triangle_func [| (expr builder e1); (expr builder
      e2); (expr builder e3);
  (expr builder e4); (expr builder e5); (expr builder e6);
  (expr builder e7); (expr builder e8)|]
  "add_triangle" builder
| SCall ("add_rectangle", [e1; e2; e3; e4; e5; e6; e7; e8]) ->
  L.build_call add_rectangle_func [| (expr builder e1); (expr
      builder e2); (expr builder e3);
  (expr builder e4); (expr builder e5); (expr builder e6);
  (expr builder e7); (expr builder e8)|]
  "add_rectangle" builder
| SCall ("add_regagon", [e1; e2; e3; e4; e5; e6; e7; e8]) ->
  L.build_call add_regagon_func [| (expr builder e1); (expr builder
      e2); (expr builder e3);
  (expr builder e4); (expr builder e5); (expr builder e6);
  (expr builder e7); (expr builder e8)|]
  "add_regagon" builder
| SCall ("add_ellipse", [e1; e2; e3; e4; e5; e6; e7; e8]) ->
  L.build_call add_ellipse_func [| (expr builder e1); (expr builder
      e2); (expr builder e3);
  (expr builder e4); (expr builder e5); (expr builder e6);
```

```
  (expr builder e7); (expr builder e8)|]
  "add_ellipse" builder
| SCall ("add_canvas", [e1; e2; e3; e4]) ->
  L.build_call canvas_func [| (expr builder e1); (expr builder e2);
      (expr builder e3); (expr builder e4);|]
  "add_canvas" builder
| SCall ("moveById", [e1; e2; e3; e4]) ->
  L.build_call moveById_func [| (expr builder e1); (expr builder
      e2); (expr builder e3); (expr builder e4);|]
  "moveById" builder
| SCall ("scaleById", [e1; e2; e3]) ->
  L.build_call scaleById_func [| (expr builder e1); (expr builder
      e2); (expr builder e3);|]
  "scaleById" builder
| SCall ("rotateById", [e1; e2; e3]) ->
L.build_call rotateById_func [| (expr builder e1); (expr builder
    e2); (expr builder e3);|]
"rotateById" builder
| SCall ("init_canvas", []) ->
  L.build_call init_canvas_func [| |]
  "init_canvas" builder
| SCall (f, args) ->
  let (fdef, fdecl) = StringMap.find f function_decls in
   let llargs = List.rev (List.map (expr builder) (List.rev args)) in
   let result = (match fdecl.styp with
                  A.Void -> ""
                | _ -> f ^ "_result")
  in
  L.build_call fdef (Array.of_list llargs) result builder

| _ -> raise (Failure "Error: Not implemented in codegen")

in

(* LLVM insists each basic block end with exactly one "terminator"
   instruction that transfers control. This function runs "instr
       builder"
   if the current block does not already have a terminator. Used,
   e.g., to handle the "fall off the end of the function" case. *)
let add_terminal builder instr =
  match L.block_terminator (L.insertion_block builder) with
Some _ -> ()
  | None -> ignore (instr builder) in

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

let rec stmt builder = function
```

```
SBlock sl -> List.fold_left stmt builder sl
   | SExpr e -> ignore(expr builder e); builder
   | SReturn e -> ignore(match fdecl.styp with
                         (* Special "return nothing" instr *)
                         A.Void -> L.build_ret_void builder
                        |
                         A.Int -> L.build_ret (L.const_null i32_t)
                             builder
                         (* Build return statement *)
                        | _ -> L.build_ret (expr builder e) builder );
                 builder
   | SIf (predicate, then_stmt, else_stmt) ->
      let bool_val = expr builder predicate in
 let merge_bb = L.append_block context "merge" the_function in
      let build_br_merge = L.build_br merge_bb in (* partial function
          *)

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

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

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

   | SWhile (predicate, body) ->
  let pred_bb = L.append_block context "while" the_function in
  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 = expr pred_builder predicate 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

   (* Implement for loops as while loops *)
   | SFor (e1, e2, e3, body) -> stmt builder
    ( SBlock [SExpr e1 ; SWhile (e2, SBlock [body ; SExpr e3]) ] )
 in

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

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

  List.iter build_function_body functions;
  the_module


**************************************************
parser.mly
**************************************************

/* Ocamlyacc parser for bugsy */

%{
open Ast
let fst' (fs,_,_)=fs
let snd' (_,sn,_)=sn
let trd (_,_,tr)=tr
%}

/* Declaring the tokens that we used */
%token CONSTRUCTOR CLASS NEW
%token LPAREN RPAREN LBRACE RBRACE LSQBRACKET RSQBRACKET
%token SEMI COMMA DOT
%token PLUS MINUS MULT DIV ASSIGN MODULO
%token INCREMENT DECREMENT
%token PLUSEQ MINUSEQ MULTEQ DIVEQ
%token EQ NEQ LT LEQ GT GEQ AND OR NOT
%token RETURN IF ELSE FOR WHILE
%token BOOL VOID STRING NUM
%token <bool> BLIT
%token <string> ID NUMLIT STRLIT CLASSID
%token EOF

/* Adding precedence and associativity for tokens */
%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left PLUSEQ MINUSEQ
%left MULT DIV MODULO
```

```
%left MULTEQ DIVEQ
%right NOT NEG
%nonassoc INCREMENT DECREMENT

%start program
%type <Ast.program> program

%%

/* The entry point for the whole a Bugsy file */
program:
  decls EOF { $1 }

/* Declarations made in a busgy file */
decls:
  /* nothing */ { [], [], [] }
  | decls vdecl { ($2 :: fst' $1), snd' $1, trd $1 }
  | decls fdecl { fst' $1, ($2 :: snd' $1), trd $1 }
  | decls cdecl { fst' $1, snd' $1, ( $2 :: trd $1) }

/* Class declarations */
cddecls:
  /* nothing */ { [], [], [] }
  | cddecls vdecl { ($2 :: fst' $1), snd' $1, trd $1 }
  | cddecls const_decl { fst' $1, ($2 :: snd' $1), trd $1 }
  | cddecls fdecl { fst' $1, snd' $1, ($2 :: trd $1) }

/* Attributes for a class including its name, constructor, attributes,
    and functions */
cdecl:
  CLASS ID LBRACE cddecls RBRACE
  { {
    cname = $2;
    cdvars = fst' $4;
    cdconst = snd' $4;
    cdfuncs = trd $4;
  } }

/* Constructor consists of formals, locally declared vars, and the body
    */
const_decl:
  CONSTRUCTOR LPAREN formals_opt RPAREN LBRACE body RBRACE
  { {
    ctformals = List.rev $3;
    ctlocals = List.rev (fst $6);
    ctbody = List.rev (snd $6);
  } }

/* Function declaration has a return type, name, parameters, and the
    function body */
```

```
fdecl:
   typ ID LPAREN formals_opt RPAREN LBRACE body RBRACE
  { {
    typ = $1;
     fname = $2;
     formals = List.rev $4;
     locals = List.rev (fst $7);
    fbody = List.rev (snd $7);
  } }

/* Body of a function consists of variable declarations and then
    statements */
body:
    /* nothing */ { [], [] }
  | body vdecl { (($2 :: fst $1), snd $1) }
  | body stmt { (fst $1, ($2 :: snd $1)) }

/* The formal arguments that a function takes */
formals_opt:
    /* nothing */ { [] }
  | formal_list  { $1 }

/* The Ocaml list of formal arguments */
formal_list:
    typ ID                { [($1,$2)]    }
  | formal_list COMMA typ ID { ($3,$4) :: $1 }

/* Built in types for Bugsy such as strings and nums */
typ:
    NUM     { Num }
  | BOOL    { Bool }
  | VOID    { Void }
  | STRING  { String }
  | array_t { $1 }
  | CLASSID { Object({
               className = $1;
               instanceVars = StringMap.empty;
             })
            }

/* Bugsy array type */
array_t:
  typ LSQBRACKET expr RSQBRACKET { Array($1, $3) }

/* Declaring a variable in Bugsy example: num n; */
vdecl:
    typ ID SEMI { ($1, $2) }

/* List of statements */
stmt_list:
```

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

/* Various statments in the Bugsy language such as if statements, or
    while loops */
stmt:
    expr SEMI { Expr $1 }
  | RETURN SEMI { Return Noexpr }
  | RETURN expr SEMI { Return $2 }
  | LBRACE stmt_list RBRACE { Block(List.rev $2) }
  | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
  | IF LPAREN expr RPAREN stmt ELSE stmt { If($3, $5, $7) }
  | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
    { For($3, $5, $7, $9) }
  | WHILE LPAREN expr RPAREN stmt { While($3, $5) }

/* If the expression is optional */
expr_opt:
    /* nothing */ { Noexpr }
  | expr        { $1 }

/* Expressions in the Bugsy language */
expr:
    ID             { Id($1) }
  | CLASSID DOT ID     { Access($1, $3) }
  | literal        { $1 }
  | bool_expr      { $1 }
  | arithmetic     { $1 }
  | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
  | CLASSID DOT ID LPAREN actuals_opt RPAREN { ClassCall($1, $3, $5) }
  | NEW ID LPAREN actuals_opt RPAREN { Construct($2, $4) }
  | ID ASSIGN expr               { Assign($1, $3) }
  | ID LSQBRACKET expr RSQBRACKET ASSIGN expr      { ArrayAssign($1, $3,
      $6) }
  | ID LSQBRACKET expr RSQBRACKET { ArrayAccess($1, $3) }
  | LPAREN expr RPAREN           { $2 } /* allow parentheses in
      arithmetic */

/* Built in types like type num or string */
literal:
    NUMLIT         { NumLit($1) }
  | STRLIT         { StrLit($1) }
  | BLIT           { BoolLit($1) }
  | LSQBRACKET arr_contents RSQBRACKET { ArrayLit(List.rev $2) }

/* Generate a list of exprs that make up the array */
arr_contents:
    expr          { [$1] }
  | arr_contents COMMA expr { $3 :: $1 }
```

```
/* Logical expressions that returns boolean values */
bool_expr:
    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) }
  | NOT expr        { Unop(Not, $2) }

/* Bugsy arithmetic operations on num types */
arithmetic:
    expr PLUS  expr { Binop($1, Add, $3) }
  | expr MINUS expr { Binop($1, Sub, $3) }
  | expr MULT  expr { Binop($1, Mult, $3) }
  | expr DIV   expr { Binop($1, Div, $3) }
  | expr MODULO expr { Binop($1, Mod, $3) }
  | expr PLUSEQ expr  { Binop($1, Pluseq, $3) }
  | expr MINUSEQ expr { Binop($1, Mineq, $3) }
  | expr MULTEQ expr  { Binop($1, Multeq, $3) }
  | expr DIVEQ  expr  { Binop($1, Diveq, $3) }
  | expr INCREMENT { Crementop($1, PostInc) }
  | INCREMENT expr { Crementop($2, PreInc) }
  | expr DECREMENT { Crementop($1, PostDec) }
  | DECREMENT expr { Crementop($2, PreDec) }
  | MINUS expr %prec NEG { Unop(Neg, $2) }

/* The actual arguments (values) that you pass into the function */
actuals_opt:
    /* nothing */ { [] }
  | actuals_list { List.rev $1 }

/* The Ocaml list of the actual arguments passed into a function call */
actuals_list:
    expr                  { [$1] }
  | actuals_list COMMA expr { $3 :: $1 }


**************************************************
sast.ml
**************************************************

open Ast

type sexpr = typ * sx
and sx =
    SNumLit of string
  | SStrLit of string
```

```
  | SBoolLit of bool
  | SArrayLiteral of sexpr list * typ
  | SIntLiteral of int
  | SId of string
  | SAccess of string * string
  | SBinop of sexpr * op * sexpr
  | SUnop of uop * sexpr
  | SAssign of string * sexpr
  | SConstruct of string * sexpr list
  | ArrayAssign of string * sexpr * sexpr
  | ArrayAccess of string * sexpr
  | SCrementop of sexpr * op
  | SCall of string * sexpr list
  | SArrayAccess of string * sexpr * typ
  | SArrayAssign of string * sexpr * sexpr
  | SClassCall of string * string * sexpr list
  | SNoexpr

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

type sconstruct_decl = {
    sctformals : bind list;
    sctlocals : bind list;
    sctbody : sstmt list;
  }

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

type scdecl = {
    scname : string;
    scdvars : bind list;
    scdconst: sconstruct_decl list;
    scdfuncs: sfunc_decl list;
    }

type sprogram = bind list * sfunc_decl list * scdecl list

(* Pretty-printing functions *)
```

```
let rec string_of_sexpr (t, e) =
  "(" ^ string_of_typ t ^ " : " ^ (match e with
    SBoolLit(true) -> "true"
  | SBoolLit(false) -> "false"
  | SNumLit(l) -> l
  | SIntLiteral(l) -> string_of_int l
  | SStrLit(l) -> l
  | SId(s) -> s
  | SAccess(s1,s2) -> s1 ^ "." ^ s2
  | SCrementop(e, o) -> string_of_sexpr e ^ " " ^ string_of_op o
  | SBinop(e1, o, e2) ->
      string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_sexpr
          e2
  | SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
  | SAssign(v, e) -> v ^ " = " ^ string_of_sexpr e
  | SConstruct(a, e) -> " new " ^ a ^ "(" ^ String.concat ", " (List.map
      string_of_sexpr e) ^ ")"
  | SCall(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^ ")"
  | SClassCall(c, f, el) ->
      c ^ "." ^ f ^ "(" ^ String.concat ", " (List.map string_of_sexpr
          el) ^ ")"
  | SArrayLiteral(el, t) -> string_of_typ t ^ "[" ^ String.concat ", "
      (List.map (fun e -> string_of_sexpr e) el) ^ "]"
  | SArrayAccess(a, e, t) -> string_of_typ t ^ " " ^ a ^ "[" ^
      string_of_sexpr e ^ "]"
  | SArrayAssign(a, left, right) -> a ^ "[" ^ string_of_sexpr left ^ "]
      = " ^ string_of_sexpr right

  | SNoexpr -> ""
  | _ -> raise (Failure "something was not implemented to print...")
           ) ^ ")"

let rec add_slevel (listy, level) = match listy with
  [] -> []
  | hd::li' -> (hd,level):: add_slevel (li', level)

let rec string_of_sstmt (stmt,level) = match stmt with
    SBlock(stmts) ->
      "{\n" ^ (String.make level '\t') ^ String.concat (String.make
          level '\t') (List.map string_of_sstmt (add_level (stmts,
          level))) ^ (String.make (level-1) '\t') ^ "}\n"
  | SExpr(expr) -> string_of_sexpr expr ^ ";\n";
  | SReturn(expr) -> "return " ^ string_of_sexpr expr ^ ";\n";
  | SIf(e, s, SBlock([])) -> "if (" ^ string_of_sexpr e ^ ")" ^
      string_of_sstmt (s, level)
  | SIf(e, s1, s2) -> "if (" ^ string_of_sexpr e ^ ")" ^
      string_of_sstmt (s1, (level)) ^ "else" ^ string_of_sstmt (s2,
          (level))
```

```
  | SFor(e1, e2, e3, s) -> "for (" ^ string_of_sexpr e1 ^ " ; " ^
      string_of_sexpr e2 ^ " ; " ^
     string_of_sexpr e3 ^ ") " ^ string_of_sstmt (s, (level+1))
  | SWhile(e, s) -> "while (" ^ string_of_sexpr e ^ ") " ^
      string_of_sstmt (s, (level+1))

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


let string_of_sconst_decl sconst_decl =
  "constructor(" ^ String.concat ", " (List.map snd
      sconst_decl.sctformals) ^
  ") {\n\t\t" ^
  String.concat "\t\t" (List.map string_of_svdecl sconst_decl.sctlocals)
      ^ "\t\t" ^
  String.concat "\t\t" (List.map string_of_sstmt (add_level
      (sconst_decl.sctbody, 1))) ^ "\t" ^
  "}\n"

let string_of_sfdecl (sfdecl, level) =
  string_of_typ sfdecl.styp ^ " " ^
  sfdecl.sfname ^ "(" ^ String.concat ", " (List.map snd
      sfdecl.sformals) ^
  ") {\n" ^ (String.make (level) '\t') ^
  String.concat (String.make level '\t') (List.map string_of_svdecl
      sfdecl.slocals) ^ (String.make level '\t') ^
  String.concat (String.make level '\t') (List.map string_of_sstmt
      (add_level (sfdecl.sfbody, (level+1)))) ^ (String.make (if
      level-1 < 0 then 0 else level-1) '\t') ^
  "}\n"

let string_of_scdecl (scdecl, level) =
  "class " ^ scdecl.scname ^ " {" ^ "\n" ^ if (List.length
      scdecl.scdvars) < 0 then "" else "\t" ^
  String.concat "\t" (List.map string_of_svdecl scdecl.scdvars) ^ "\t" ^
  String.concat "\t" (List.map string_of_sconst_decl scdecl.scdconst) ^
      "\t" ^
  String.concat "\t" (List.map string_of_sfdecl (add_level
      (scdecl.scdfuncs, (level+1)))) ^ (String.make (level-1) '\t') ^
  "}\n"

let string_of_sprogram (svars, sfuncs, sclasses) =
  String.concat "" (List.map string_of_svdecl svars) ^ "\n" ^
  String.concat "" (List.map string_of_scdecl (add_level (sclasses, 1)))
      ^ "\n" ^
  String.concat "\n" (List.map string_of_sfdecl (add_level (sfuncs, 1)))



**************************************************
```

```
scanner.mll
**************************************************

(* Ocamllex scanner for bugsy *)

{ open Parser }

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

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"    { comment 0 lexbuf }        (* Comments *)
| "//"    { linecomment lexbuf }    (* single line comments *)

(* Symbols *)
| '('     { LPAREN }
| ')'     { RPAREN }
| '{'     { LBRACE }
| '}'     { RBRACE }
| '['     { LSQBRACKET }
| ']'     { RSQBRACKET }
| ';'     { SEMI  }
| ','     { COMMA }
| '.'     { DOT   }

(* Arithmetic Tokens *)
| '+'     { PLUS  }
| "+="    { PLUSEQ }
| '-'     { MINUS }
| "-="    { MINUSEQ   }
| '*'     { MULT }
| '/'     { DIV  }
| "*="    { MULTEQ    }
| "/="    { DIVEQ     }
| '%'     { MODULO }
| "++"    { INCREMENT }
| "--"    { DECREMENT }
| '='     { ASSIGN }

(* Boolean Expression Tokens *)
| "=?"    { EQ    }
| "!="    { NEQ   }
| '<'     { LT    }
| "<="    { LEQ   }
| ">"     { GT    }
| ">="    { GEQ   }
| "and"   { AND   }
| "or"    { OR    }
| "!"     { NOT   }
```

```
(* Control Flow *)
| "if"    { IF    }
| "else"  { ELSE  }
| "for"   { FOR   }
| "while" { WHILE }
| "return" { RETURN }

(* Classes *)
| "class" { CLASS }
| "constructor" { CONSTRUCTOR }
| "new" { NEW }

(* Builtin Types *)
| "num"   { NUM   }
| "bool"  { BOOL  }
| "void"  { VOID  }
| "string" { STRING }
| "True" |"true" { BLIT(true) }
| "False"|"false" { BLIT(false) }

(* Numeric Literal *)
| digits '.'? digit* ( ['e' 'E'] ['+' '-']? digits )? as lxm {
    NUMLIT(lxm) }
| ['a'-'z' '_' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| '~'['a'-'z' '_' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm {
    CLASSID(lxm) }

(* String Literal *)
|'\"' ([^ '\"' '\n' '\r' '\t' '\b']* as lxm) '\"' { STRLIT(lxm) }

| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

and comment nest_level = parse
  "*/" { match nest_level with
           0 -> token lexbuf
         | _ -> comment (nest_level - 1) lexbuf
       }
| "/*" { comment (nest_level + 1) lexbuf }
| _    { comment nest_level lexbuf }

and linecomment = parse
  "\n" { token lexbuf }
| eof  { token lexbuf }
| _    { linecomment lexbuf }


**************************************************
semant.ml
```

```
*************************************************

(* Semantic checking for the MicroC compiler *)

open Ast
open Sast

module StringMap = Map.Make(String)


let rec zip_list (listy, li2) = match listy with
    [] -> []
  | hd::li' -> (hd, li2)::zip_list(li', li2)

(* let rec unzip' (tup) = match tup with
    [] -> []
  | (elem, li)::li' -> elem::unzip'(li')

let unzip tup = match tup with
    [] -> ([], [])
  | (elem, li)::li' -> (elem::unzip'(li'), li) *)

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

let check (globals, functions, classes) =

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

  (* Add Class Helper *)
  let add_class map cd =
    (*let built_in_err = "class " ^ cd.cname ^ " may not be defined"*)
    let dup_err = "duplicate class " ^ cd.cname
    and make_err er = raise (Failure er)
    and n = cd.cname (* Name of the class *)
    in match cd with (* No duplicate classes or redefinitions of
        built-ins *)
       _ when StringMap.mem n map -> make_err dup_err
```

```
    | _ -> StringMap.add n cd map
in


(* Helper function for checking if a class is defined *)
let verify_class_name cname =
  (* Collect all class names into one symbol table *)
  (* TODO: Fix the unused class_decls here *)
  let class_decls = List.fold_left add_class (*built_in_class_decls*)
      StringMap.empty classes
  in
  let find_class s =
    try StringMap.find s class_decls
    with Not_found -> raise (Failure ("unrecognized class " ^ s))
  in
  find_class cname
in

(* Find class *)
let get_class cname =
  let class_decls = List.fold_left add_class StringMap.empty classes
  in
  let find_class s =
    try StringMap.find s class_decls
    with Not_found -> raise (Failure ("unrecognized class " ^ s))
  in
  find_class cname (* Return the class from its name *)
in

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

check_binds "global" globals;

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

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

let built_in_decls =
  let add_bind map (name, ty) = StringMap.add name {
    typ = Void;
    fname = name;
    formals = [(ty, "x")];
    locals = []; fbody = [] } map
  in let map1 = List.fold_left add_bind StringMap.empty [
                               ("printb", Bool);
                               ("print", Num);
                                  ("printf", String);
                               ("printbig", Num);]

  in
```

```
let add_bind2 map (name) = StringMap.add name {
  typ = Void;
  fname = name;
  formals = [];
  locals = []; fbody = [] } map
in let map2 = List.fold_left add_bind2 map1 [
                                ("demo");]
in
let add_bind4 map (name, ty1, ty2, ty3) = StringMap.add name {
  typ = Void;
  fname = name;
  formals = [(ty1, "x"); (ty2, "y"); (ty3, "id")];
  locals = []; fbody = [] } map
in let map4 = List.fold_left add_bind4 map2 [
                     ("add_point_xy", Num, Num, String);]
in
let add_bind5 map (name, ty1, ty2, ty3, ty4, ty5, ty6, ty7) =
    StringMap.add name {
  typ = String;
  fname = name;
  formals = [(ty1, "x"); (ty2, "y"); (ty3, "r"); (ty4, "stroke");
      (ty5, "thickness"); (ty6, "fill"); (ty7, "id")];
  locals = []; fbody = [] } map
in let map5 = List.fold_left add_bind5 map4 [
                     ("add_circle", Num, Num, Num, String, Num,
                         String, String);]
in
let add_bind6 map (name, ty1, ty2, ty3, ty4, ty5, ty6, ty7) =
    StringMap.add name {
  typ = String;
  fname = name;
  formals = [(ty1, "x"); (ty2, "y"); (ty3, "size"); (ty4, "stroke");
      (ty5, "thickness"); (ty6, "fill"); (ty7, "id")];
  locals = []; fbody = [] } map
in let map6 = List.fold_left add_bind6 map5 [
                     ("add_square", Num, Num, Num, String, Num,
                         String, String);]
in
let add_bind7 map (name, ty1, ty2, ty3, ty4) = StringMap.add name {
  typ = Void;
  fname = name;
  formals = [(ty1, "id"); (ty2, "translateX"); (ty3, "translateY");
      (ty4, "speed")];
  locals = []; fbody = [] } map
in let map7 = List.fold_left add_bind7 map6 [
                     ("moveById", String, Num, Num, Num);]
in
let add_bind8 map (name, ty1, ty2, ty3, ty4) = StringMap.add name {
  typ = Void;
  fname = name;
```

```
  formals = [(ty1, "width"); (ty2, "height"); (ty3, "xOffset");
      (ty4, "yOffset")];
  locals = []; fbody = [] } map
in let map8 = List.fold_left add_bind8 map7 [
                      ("add_canvas", Num, Num, Num, Num);]
in
let add_bind9 map (name, ty1, ty2, ty3) = StringMap.add name {
  typ = Void;
  fname = name;
  formals = [(ty1, "id"); (ty2, "angle"); (ty3, "speed");];
  locals = []; fbody = [] } map
in let map9 = List.fold_left add_bind9 map8 [
                      ("rotateById", String, Num, Num);]
in
let add_bind10 map (name, ty1, ty2, ty3) = StringMap.add name {
  typ = Void;
  fname = name;
  formals = [(ty1, "id"); (ty2, "scale"); (ty3, "speed")];
  locals = []; fbody = [] } map
in let map10 = List.fold_left add_bind10 map9 [
                      ("scaleById", String, Num, Num);]
in
let add_bind11 map (name, ty1, ty2, ty3, ty4, ty5, ty6, ty7, ty8) =
    StringMap.add name {
  typ = String;
  fname = name;
  formals = [(ty1, "x"); (ty2, "y"); (ty3, "b"); (ty4, "h"); (ty5,
      "stroke"); (ty6, "thickness"); (ty7, "fill"); (ty8, "id")];
  locals = []; fbody = [] } map
in let map11 = List.fold_left add_bind11 map10 [
                      ("add_triangle", Num, Num, Num, Num, String,
                          Num, String, String);]
in
let add_bind12 map (name, ty1, ty2, ty3, ty4, ty5, ty6, ty7, ty8) =
    StringMap.add name {
  typ = String;
  fname = name;
  formals = [(ty1, "x"); (ty2, "y"); (ty3, "w"); (ty4, "h"); (ty5,
      "stroke"); (ty6, "thickness"); (ty7, "fill"); (ty8, "id")];
  locals = []; fbody = [] } map
in let map12 = List.fold_left add_bind12 map11 [
                      ("add_rectangle", Num, Num, Num, Num, String,
                          Num, String, String);]
in
let add_bind13 map (name, ty1, ty2, ty3, ty4, ty5, ty6, ty7, ty8) =
    StringMap.add name {
  typ = String;
  fname = name;
  formals = [(ty1, "x"); (ty2, "y"); (ty3, "w"); (ty4, "h"); (ty5,
      "stroke"); (ty6, "thickness"); (ty7, "fill"); (ty8, "id")];
```

```
    locals = []; fbody = [] } map
  in let map13 = List.fold_left add_bind13 map12 [
                       ("add_ellipse", Num, Num, Num, Num, String,
                           Num, String, String);]
  in
  let add_bind14 map (name, ty1, ty2, ty3, ty4, ty5, ty6, ty7, ty8) =
      StringMap.add name {
    typ = String;
    fname = name;
    formals = [(ty1, "x"); (ty2, "y"); (ty3, "n"); (ty4, "r"); (ty5,
        "stroke"); (ty6, "thickness"); (ty7, "fill"); (ty8, "id")];
    locals = []; fbody = [] } map
  in let map14 = List.fold_left add_bind14 map13 [
                       ("add_regagon", Num, Num, Num, Num, String,
                           Num, String, String);]
  in
  let add_bind15 map (name) = StringMap.add name {
    typ = Void;
    fname = name;
    formals = [];
    locals = []; fbody = [] } map
  in let map15 = List.fold_left add_bind15 map14 [
                       ("init_canvas");]
  in
  let add_bind16 map (name, ty1, ty2, ty3, ty4, ty5, ty6, ty7) =
      StringMap.add name {
    typ = String;
    fname = name;
    formals = [(ty1, "x1"); (ty2, "y1"); (ty3, "x2"); (ty4, "y2");
        (ty5, "stroke"); (ty6, "thickness"); (ty7, "id")];
    locals = []; fbody = [] } map
  in List.fold_left add_bind16 map15 [
                       ("add_line", Num, Num, Num, Num, String, Num,
                           String);]
in
(* Add function name to symbol table *)
let add_func map fd =
  let built_in_err = "function " ^ fd.fname ^ " may not be defined"
  and dup_err = "duplicate function " ^ fd.fname
  and make_err er = raise (Failure er)
  and n = fd.fname (* Name of the function *)
  in match fd with (* No duplicate functions or redefinitions of
      built-ins *)
       _ when StringMap.mem n built_in_decls -> make_err built_in_err
     | _ when StringMap.mem n map -> make_err dup_err
     | _ -> StringMap.add n fd map
in

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

```
(*let function_decls = List.fold_left add_func functions*)
in

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

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

let check_function tup =
  let func = fst tup in (* function declaration *)
  let opt_class = snd tup in (* Class binds if the function is nested
      in a class *)

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

  (* Raise an exception if the given rvalue type cannot be assigned to
     the given lvalue type *)
  let rec check_assign lvaluet rvaluet err =  (* haww *)
      (* get class name *)
    let className = match lvaluet with
    (* if it's an object, pull out the class name *)
      Object(o) -> (match o with
            {className; _} -> className
          )
      | _ -> ""
    in
    (* check if there is a class *)
    match className with

    "" -> ( match rvaluet with
     Array(t1, _) -> (match t1 with
          Num -> check_assign t1 Num err
          | String -> check_assign t1 String err
          | _ -> raise (Failure err))
       | _ -> if lvaluet = rvaluet then lvaluet else raise (Failure
           err))

    | s -> if not (String.equal s
      (
      match rvaluet with
        Object(o) -> (match o with
             {className; _} -> className)
        | _ -> ""

        (*if not, then raise an err, otherwise return lvalue*)
```

```
    )) then raise (Failure err) else lvaluet
in

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

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

let access_type = function
       Array(t, _) -> t
   | _ -> raise (Failure("illegal array access"))
in

(* Return a semantically-checked expression, i.e., with a type *)
let rec expr = function
        NumLit l  -> (Num, SNumLit l)
      | ArrayLit l ->

    let test = string_of_int (List.length l) in
    (Array (Num, NumLit(test)), SArrayLiteral(List.map expr l,
        Array(Num, IntLiteral(List.length l))))

  | ArrayAccess(a, e) -> check_int e; (type_of_identifier a,
      SArrayAccess(a, expr e, access_type (type_of_identifier a)))
  | ArrayAssign(var, idx, num) -> check_int num; check_int idx;
      (type_of_identifier var, SArrayAssign(var, expr idx, expr num))

  | IntLiteral l -> (Num, SIntLiteral l)
  | BoolLit l -> (Bool, SBoolLit l)
  | StrLit l  -> (String, SStrLit l)
  | Noexpr    -> (Void, SNoexpr)
  | Id s      -> (type_of_identifier s, SId s)
  | Access (obj, var) ->
     let ctyp = string_of_typ (type_of_identifier obj) in (* Get
         class name from object name *)
     let _ = verify_class_name ctyp in (*make sure class exists *)
     let class_object = get_class ctyp in (*get the class we need to
         check *)

     (* Build local symbol table of variables for this function *)
     let classSymbols = List.fold_left (fun m (ty, name) ->
         StringMap.add name ty m)
```

```
    StringMap.empty (class_object.cdvars)
    in
    (* Find the variable in the class *)
    let find_var s =
      try StringMap.find s classSymbols
      with Not_found -> raise (Failure ("unrecognized variable " ^
          s))
    in
    let cv = find_var var in
    (cv, SAccess(obj, var))
| Assign(var, e) as ex ->
    let lt = type_of_identifier var
    and (rt, e') = expr e in
    let err = "illegal assignment v1 " ^ string_of_typ lt ^ " = " ^
      string_of_typ rt ^ " in " ^ string_of_expr ex
    in (check_assign lt rt err, SAssign(var, (rt, e')))
| Construct(cn, args) as construct ->
    (* Add class name to symbol table *)
    let add_class map cd =
      let dup_err = "duplicate class " ^ cd.cname
      and make_err er = raise (Failure er)
      and n = cd.cname (* Name of the class *)
      in match cd with (* No duplicate classes or redefinitions of
          built-ins *)
        | _ when StringMap.mem n map -> make_err dup_err
        | _ -> StringMap.add n cd map
    in

    (* Collect all class names into one symbol table *)
    (* TODO: Fix the unused class_decls here *)
    let class_decls = List.fold_left add_class StringMap.empty
        classes
    in

    let find_class s =
      try StringMap.find s class_decls
      with Not_found -> raise (Failure ("unrecognized class " ^ s))
    in

    let _class = find_class cn in (* Ensure class cn is defined *)
    (*_class.ctformals *)
    let ct = List.nth _class.cdconst 0 in
    let formals = ct.ctformals in
    let param_length = List.length formals in
    if List.length args != param_length then
      raise (Failure ("expecting " ^ string_of_int param_length ^
                    " arguments in " ^ string_of_expr construct ))
    else let check_const_call (formal_typ,_) e =
      let (et, e') = expr e in
      let err = "illegal argument found " ^ string_of_typ et ^
```

```
      " expected " ^ string_of_typ formal_typ ^ " in " ^
          string_of_expr e
    in (check_assign formal_typ et err, e')
  in
  let args' = List.map2 check_const_call formals args in
  let vars = List.fold_left2
    (fun map (ty, n) value-> StringMap.add n (ty,value) map)
    StringMap.empty formals args
  in (Object({className = cn; instanceVars = vars ;}) ,
      SConstruct(cn, args'))

| Unop(op, e) as ex ->
  let (t, e') = expr e in
  let ty = match op with
 (*   Neg when t = Num || t = Float -> t *)
    Neg when t = Num -> t
  | Not when t = Bool -> Bool
  | _ -> raise (Failure ("illegal unary operator " ^
                         string_of_uop op ^ string_of_typ t ^
                         " in " ^ string_of_expr ex))
  in (ty, SUnop(op, (t, e')))
| Crementop(e, op) as ex ->
  let (t, e') = expr e in
  let ty = match op with
    PreInc when t = Num -> Num
  | PostInc when t = Num -> Num
  | PreDec when t = Num -> Num
  | PostDec when t = Num -> Num
  | _ -> raise (Failure ("illegal increment/decrement operator " ^
                         string_of_op op ^ string_of_typ t ^
                         " in " ^ string_of_expr ex))
  in (ty, SCrementop((t, e'), op))
| Binop(e1, op, e2) as e ->
  let (t1, e1') = expr e1
  and (t2, e2') = expr e2 in
  (* All binary operators require operands of the same type *)
  let same = t1 = t2 in
  (* Determine expression type based on operator and operand
      types *)
  let ty = match op with
    Add | Sub | Mult | Div when same && t1 = Num -> Num
 (* | Add | Sub | Mult | Div when same && t1 = Float -> Float *)
  | Equal | Neq          when same             -> Bool
 (*  | Less | Leq | Greater | Geq
            when same && (t1 = Num || t1 = Float) -> Bool *)
  |  Less | Leq | Greater | Geq
              when same && (t1 = Num) -> Bool
  | And | Or when same && t1 = Bool -> Bool
  | _ -> raise (
 Failure ("illegal binary operator " ^
```

```
                    string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
                    string_of_typ t2 ^ " in " ^ string_of_expr e))
        in (ty, SBinop((t1, e1'), op, (t2, e2')))
    | Call(fname, args) as call ->
        let fd = find_func fname in
        let param_length = List.length fd.formals in
        if List.length args != param_length then
          raise (Failure ("expecting " ^ string_of_int param_length ^
                          " arguments in " ^ string_of_expr call))
        else let check_call (ft, _) e =
          let (et, e') = expr e in
          let err = "illegal argument found " ^ string_of_typ et ^
            " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr e
          in (check_assign ft et err, e')
        in
        let args' = List.map2 check_call fd.formals args
        in (fd.typ, SCall(fname, args'))

      | ClassCall(cname, fname, args) as classcall ->
        let ctyp = string_of_typ (type_of_identifier cname) in (* Get
            class name from object name *)
        let _ = verify_class_name ctyp in
        let class_object = get_class ctyp in
        let cfuncs = List.fold_left add_func StringMap.empty
            class_object.cdfuncs in

        (* Find the function in the class *)
        let find_func s =
          try StringMap.find s cfuncs
          with Not_found -> raise (Failure ("unrecognized function " ^
              cname ^ "." ^ fname))
        in
        let cf = find_func fname in
        let param_length = List.length cf.formals in
        if List.length args != param_length then
          raise (Failure ("expecting " ^ string_of_int param_length ^
                          " arguments in " ^ string_of_expr classcall))
        else let check_call (ft, _) e =
          let (et, e') = expr e in
          let err = "illegal argument found " ^ string_of_typ et ^
            " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr e
          in (check_assign ft et err, e')
        in
        let args' = List.map2 check_call cf.formals args
        in (cf.typ, SClassCall(cname, fname, args'))
and check_int e =
        let (t', e') = expr e
        and err = "expected Int expression in " ^ string_of_expr e
        in if t' != Num then raise (Failure err) else ignore e'
```

```
 in

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

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

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

 in (* body of check_function *)
 { styp = func.typ;
   sfname = func.fname;
   sformals = func.formals;
   slocals = func.locals;
   sfbody = match check_stmt (Block func.fbody) with
SBlock(sl) -> sl
   | _ -> raise (Failure ("internal error: block didn't become a
       block?"))
 }
in

(**** Check Classes ****)

(* Collect class declarations for built-in classes: no bodies *)
```

```
(*
  let built_in_class_decls =
      let add_bind map (name, _, _) = StringMap.add name {
          cname = name;
          cdvars = [];
          cdconst = []; cdfuncs = [] } map
      in List.fold_left add_bind StringMap.empty []

      in*)
    (*  let add_bind2 map (name) = StringMap.add name {
          cname = name;
          cdvars = [];
          cdconst = []; cdfuncs = [] } map
      in List.fold_left add_bind2 func_map []*)

(*
  (* Add class name to symbol table *)
  let add_class map cd =
    let built_in_err = "class " ^ cd.cname ^ " may not be defined"
    and dup_err = "duplicate class " ^ cd.cname
    and make_err er = raise (Failure er)
    and n = cd.cname (* Name of the class *)
    in match cd with (* No duplicate classes or redefinitions of
        built-ins *)
        _ when StringMap.mem n built_in_class_decls -> make_err
            built_in_err
      | _ when StringMap.mem n map -> make_err dup_err
      | _ -> StringMap.add n cd map
  in
*)
  (* Collect all class names into one symbol table *)
  (* TODO: Fix the unused class_decls here *)
  (*let class_decls = List.fold_left add_class built_in_class_decls
      classes
  in*)

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

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

  let check_class _class =
    (* Make sure no formals or locals are void or duplicates *)
    check_binds "cdvar" _class.cdvars;

(*
    (* Raise an exception if the given rvalue type cannot be assigned to
```

```
      the given lvalue type *)
    let check_assign lvaluet rvaluet err =
      if lvaluet = rvaluet then lvaluet else raise (Failure err)
    in
*)
    (* Build local symbol table of variables for this function *)
    let class_symbols = List.fold_left (fun m (ty, name) ->
        StringMap.add name ty m)
                  StringMap.empty (globals @ _class.cdvars )
    in
(*
    (* Return a variable from our local symbol table *)
    let type_of_identifier s =
      try StringMap.find s class_symbols
      with Not_found -> raise (Failure ("undeclared identifier " ^ s))
    in

    (* Return a semantically-checked expression, i.e., with a type *)
    let rec expr = function
        NumLit l  -> (Num, SNumLit l)
      | BoolLit l -> (Bool, SBoolLit l)
      | StrLit l  -> (String, SStrLit l)
      | Noexpr    -> (Void, SNoexpr)
      | Id s      -> (type_of_identifier s, SId s)
      | Access(obj, var) ->
          let ctyp = string_of_typ (type_of_identifier obj) in (* Get
              class name from object name *)
          let _ = verify_class_name ctyp in (*make sure class exists *)
          let _ = get_class ctyp in (*get the class we need to check *)
          let vt = type_of_identifier var in
          (vt, SAccess(obj, var))

      | Assign(var, e) as ex ->
          let lt = type_of_identifier var
          and (rt, e') = expr e in
          let err = "illegal assignment v2 " ^ string_of_typ lt ^ " = " ^
            string_of_typ rt ^ " in " ^ string_of_expr ex
          in (check_assign lt rt err, SAssign(var, (rt, e')))
      | Unop(op, e) as ex ->
          let (t, e') = expr e in
          let ty = match op with
        (*    Neg when t = Num || t = Float -> t *)
            Neg when t = Num -> t
          | Not when t = Bool -> Bool
          | _ -> raise (Failure ("illegal unary operator " ^
                                 string_of_uop op ^ string_of_typ t ^
                                 " in " ^ string_of_expr ex))
          in (ty, SUnop(op, (t, e')))
      | Binop(e1, op, e2) as e ->
          let (t1, e1') = expr e1
```

```
        and (t2, e2') = expr e2 in
        (* All binary operators require operands of the same type *)
        let same = t1 = t2 in
        (* Determine expression type based on operator and operand
            types *)
        let ty = match op with
          Add | Sub | Mult | Div when same && t1 = Num -> Num
        (* | Add | Sub | Mult | Div when same && t1 = Float -> Float *)
        | Equal | Neq          when same             -> Bool
       (*  | Less | Leq | Greater | Geq
                  when same && (t1 = Num || t1 = Float) -> Bool *)
        |  Less | Leq | Greater | Geq
                    when same && (t1 = Num) -> Bool
        | And | Or when same && t1 = Bool -> Bool
        | _ -> raise (
       Failure ("illegal binary operator " ^
                    string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
                    string_of_typ t2 ^ " in " ^ string_of_expr e))
        in (ty, SBinop((t1, e1'), op, (t2, e2')))
    | _ -> raise (Failure "Not implemented")
  in
*)
  (*
  let check_bool_expr e =
    let (t', e') = expr e
    and err = "expected Boolean expression in " ^ string_of_expr e
    in if t' != Bool then raise (Failure err) else (t', e')
  in
  *)

  (**** Check Constructors ****)

  (* Collect function declarations for built-in functions: no bodies *)
(*
  let built_in_const_decls =
    let add_bind map (name, _) = StringMap.add name {
      ctformals = [];
      ctlocals = []; ctbody = [] } map
    in List.fold_left add_bind StringMap.empty []
  in
*)
  (*
  (* Add function name to symbol table *)
  let add_const map ctd =
    let built_in_err = "constructor may not be defined"
    and dup_err = "duplicate constructor"
    and make_err er = raise (Failure er)
    and n = ""
    in match ctd with (* No duplicate functions or redefinitions of
        built-ins *)
```

```
          _ when StringMap.mem n built_in_const_decls -> make_err
              built_in_err
        | _ when StringMap.mem n map -> make_err dup_err
        | _ -> StringMap.add n ctd map
    in
*)
    (* Collect all function names into one symbol table *)
    (*let constructor_decls = List.fold_left add_const
        built_in_const_decls _class.cdconst
    (*let function_decls = List.fold_left add_func functions*)
    in*)

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

    let check_constructor const =
      (* Make sure no formals or locals are void or duplicates *)
      check_binds "ctformal" const.ctformals;
      check_binds "ctlocal" const.ctlocals;

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

      (* Build local symbol table of variables for this function *)
      let local_symbols = List.fold_left (fun m (ty, name) ->
          StringMap.add name ty m)
                    StringMap.empty (globals @ const.ctformals @
                        const.ctlocals )
      in

      (* Return a variable from our local symbol table *)
      let type_of_identifier s =
        try StringMap.find s (StringMap.merge (fun _ xo yo -> match
            xo,yo with
        | Some x, Some _ -> Some (x)
        | None, yo -> yo
        | xo, None -> xo
        ) class_symbols local_symbols)
        with Not_found -> raise (Failure ("undeclared identifier " ^ s))
      in

 let access_type = function
           Array(t, _) -> t
```

```
   | _ -> raise (Failure("illegal array access"))
in


  (* Return a semantically-checked expression, i.e., with a type *)
  let rec expr = function
    | ArrayLit l ->
       (Array (Num, IntLiteral(List.length l)),
       SArrayLiteral(List.map expr l,
       Array(Num, IntLiteral(List.length l))))

    | ArrayAccess(a, e) ->
       check_int e;
       (type_of_identifier a,
       SArrayAccess(a, expr e, access_type (type_of_identifier a)))

    | IntLiteral l -> (Num, SIntLiteral l)
    | NumLit l  -> (Num, SNumLit l)
    | BoolLit l -> (Bool, SBoolLit l)
    | StrLit l  -> (String, SStrLit l)
    | Noexpr    -> (Void, SNoexpr)
    | Id s      -> (type_of_identifier s, SId s)
    | Access (s1, s2) -> (type_of_identifier s2, SAccess(s1, s2))
    | Assign(var, e) as ex ->
       let lt = type_of_identifier var
       and (rt, e') = expr e in
       let err = "illegal assignment v3 " ^ string_of_typ lt ^ " = "
           ^
         string_of_typ rt ^ " in " ^ string_of_expr ex
       in (check_assign lt rt err, SAssign(var, (rt, e')))
    | Unop(op, e) as ex ->
       let (t, e') = expr e in
       let ty = match op with
      (*    Neg when t = Num || t = Float -> t *)
          Neg when t = Num -> t
       | Not when t = Bool -> Bool
       | _ -> raise (Failure ("illegal unary operator " ^
                              string_of_uop op ^ string_of_typ t ^
                              " in " ^ string_of_expr ex))
       in (ty, SUnop(op, (t, e')))
    | Crementop(e, op) as ex ->
      let (t, e') = expr e in
      let ty = match op with
        PreInc when t = Num -> Num
      | PostInc when t = Num -> Num
      | PreDec when t = Num -> Num
      | PostDec when t = Num -> Num
      | _ -> raise (Failure ("illegal increment/decrement operator " ^
                             string_of_op op ^ string_of_typ t ^
```

```
                        " in " ^ string_of_expr ex))
    in (ty, SCrementop((t, e'), op))
  | Binop(e1, op, e2) as e ->
     let (t1, e1') = expr e1
     and (t2, e2') = expr e2 in
     (* All binary operators require operands of the same type *)
     let same = t1 = t2 in
     (* Determine expression type based on operator and operand
         types *)
     let ty = match op with
       Add | Sub | Mult | Div when same && t1 = Num -> Num
    (* | Add | Sub | Mult | Div when same && t1 = Float -> Float *)
     | Equal | Neq          when same             -> Bool
   (*  | Less | Leq | Greater | Geq
              when same && (t1 = Num || t1 = Float) -> Bool *)
     |  Less | Leq | Greater | Geq
                when same && (t1 = Num) -> Bool
     | And | Or when same && t1 = Bool -> Bool
     | _ -> raise (
     Failure ("illegal binary operator " ^
                string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
                string_of_typ t2 ^ " in " ^ string_of_expr e))
     in (ty, SBinop((t1, e1'), op, (t2, e2')))
  | _ -> raise (Failure "Error: Semant: Not Implemented")

and check_int e =
   let (t', e') = expr e
   and err = "expected Int expression in " ^ string_of_expr e
   in if t' != Num then raise (Failure err) else ignore e'

in


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

(* Return a semantically-checked statement i.e. containing sexprs
    *)
let rec check_stmt = function
    Expr e -> SExpr (expr e)
  | If(p, b1, b2) -> SIf(check_bool_expr p, check_stmt b1,
      check_stmt b2)
  | For(e1, e2, e3, st) ->
  SFor(expr e1, check_bool_expr e2, expr e3, check_stmt st)
  | While(p, s) -> SWhile(check_bool_expr p, check_stmt s)

    (* A block is correct if each statement is correct and nothing
```

```
           follows any Return statement. Nested blocks are flattened. *)
        | Block sl ->
           let rec check_stmt_list = function
               [Return _ as s] -> [check_stmt s]
             | Return _ :: _ -> raise (Failure "nothing may follow a
                 return")
             | Block sl :: ss -> check_stmt_list (sl @ ss) (* Flatten
                 blocks *)
             | s :: ss       -> check_stmt s :: check_stmt_list ss
             | []            -> []
           in SBlock(check_stmt_list sl)
        | _ -> raise (Failure ("Not semantically valid. You might've
            returned in a constructor."))

      in (* body of check_function *)
      {
        sctformals = const.ctformals;
        sctlocals = const.ctlocals;
        sctbody = match check_stmt (Block const.ctbody) with
      SBlock(sl) -> sl
        | _ -> raise (Failure ("internal error: block didn't become a
            block?"))
      }
    in ignore(globals, List.map check_constructor _class.cdconst);

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

    {
      scname = _class.cname;
      scdvars = _class.cdvars;
      scdconst = List.map check_constructor _class.cdconst;
      scdfuncs = List.map check_function (zip_list (_class.cdfuncs,
          _class.cdvars));
    }
  in (globals, List.map check_function (zip_list (functions, [])),
      List.map check_class classes)


**************************************************
**************************************************
animation-array-example.bug
**************************************************

num main() {
    num cx;
    num cy;
    num scale;
    num x;
    num y;
    num cw;
```

```
    num ch;
    num r;
    num w;
    num h;
    num s;
    num n;
    num b;
    cx = 0;
    cy = 0;
    cw = 1500;
    ch = 1000;
    x = cw / 2;
    y = ch / 2;


    string stroke;
    string cir_fill;

    stroke = "0.0 0.3 0.5";
    cir_fill = "1.0 0.2 0.6";

    string id;
    id = "";

    init_canvas();

    num i;
    for(i = 0; i < 10; i++) {
        id = add_circle(x+2000, y+2000, 50 + (i * 10), stroke, 5,
            cir_fill, id);
        moveById(id, -2000, -2000, 1);
    }

    add_canvas(cw, ch, cx, cy);
}


**************************************************
array_demo.bug
**************************************************

num main(){
    num[5] x;
    num i;
    num j;
    num var;
    num test;

    test = 5.4;
```

```
    test = 1.0;

    x = [5.4, 3.4, 6.7, 5.1, 0.6];


    printf("The original values of the array:");
    printf("");
    for (i = 0; i < 5; i++){
        printf("element");
        print(i);
        printf("is");
        print(x[i]);
    }

    printf("Looping through and changing up array values...");
    printf("");
    printf("");
    i = 0;
    for (j = 0; j < 5; j++){
        x[j] = j * 2;
    }


    printf("The updated values of the array:");
    printf("");


    for (i = 0; i < 5; i++){
        printf("element");
        print(i);
        printf("is");
        print(x[i]);
    }


    return 0;
}


**************************************************
arrays.bug
**************************************************

num main() {

   num[5] x;
   num a;
   num b;
   num c;
   num d;
```

```
  num e;
  x = [10.9, 30.6, 10.3, 5.4, 6.7];

  printf("The array should be [10.9, 30.6, 10.3, 5.4, 6.7]");
  printf("The first element is");
  print(x[0]);


  a = 2.1;

  b = 2;

  e = x[b];

  e = 4.0;

  x[2] = a;

  c = 3.4;
  x[3] = c;


  x[4] = 9.0;

  printf("sirs second element is");
  printf("sirs third element is");
  print(x[3]);
  printf("sirs fourth element is");
  print(x[4]);


  for (c = 0; c < 4; c++){
   printf("element number");
   print(c);
   printf("is");
   d  = x[e];
   print(d);
 }



}


*****************************************************
fact.bug
*****************************************************

num fact(num n){
```

```
    if (n =? 1) {
        return n;
    }

    else {

    return n * fact(n-1);

    }
    }

num main(){
    num a;
    a = fact(4);
   print(a);


    return 0;
    }


**************************************************
fizzbug.bug
**************************************************

  num modulus(num a, num b){
       num ret;
       num mod;

       if (a < 0){

          a = -1 * a;
        }

        if (b < 0){

          b = -1 * b;

        }

        mod = a;

        while (mod >= b){
            mod = mod - b;

        }

        if (a < 0){
```

```
            return -1 * mod;
            }

        return mod;


        }


num main(){


    num n;
    num i;

    n = 100;

      for (i = 1; i <= n; i++){
        if (modulus(i, 15) =? 0){
            printf("FizzBug");

            }
        else {

        if (modulus(i, 5) =? 0){
            printf("Bug");
            }

        else
        {
            if (modulus(i, 3) =? 0){
            printf("Fizz");
            }

        else {
            print(i);
            }
        }
        }
      }

    return 0;
    }


**************************************************
loop.bug
**************************************************

num main(){
```

```
  num x;
  num y[5];
  x = 0;
  y = [1, 2, 3, 4, 5];

  for (x = 0; x < 5; x = x + 1){
     print(0);

  }

  return 0;

}


**************************************************
shape_test.bug
**************************************************

num main() {
    num cx;
    num cy;
    num scale;
    num x;
    num y;
    num cw;
    num ch;
    num r;
    num w;
    num h;
    num s;
    num n;
    num b;
    cx = 0;
    cy = 0;
    cw = 1500;
    scale = 3;
    ch = 1000;
    x = cw / 2;
    y = ch / 2;
    r = 100;
    s = 75;
    n = 10;
    b = 60;
    w = 100;
    h = 100;

    string stroke;
    string reg_fill;
```

```
    string tri_fill;
    string rect_fill;
    string sq_fill;
    string cir_fill;
    string ell_fill;

    stroke = "0.0 0.3 0.5";
    reg_fill = "0.2 0.2 0.2";
    cir_fill = "1.0 0.2 0.6";
    ell_fill = "0.8 0.5 0.5";
    sq_fill = "0.9 0.9 0.2";
    tri_fill = "0.7 0.4 0.0";
    rect_fill = "1.0 0.9 0.6";

    num thickness;
    thickness = 5;
    string id;
    string id2;
    string id3;
    string id4;
    string id5;
    string id6;
    id = "";
    id2 = "";
    id3 = "";
    id4 = "";
    id5 = "";
    id6 = "";


    init_canvas();

    add_ellipse(x+300, y, w, h*2, stroke, thickness, ell_fill, id6);
    add_circle(x-300, y, r, stroke, thickness, cir_fill, id);
    add_square(x, y+300, s, stroke, thickness, sq_fill, id2);
    add_rectangle(x, y-300, w*2, h, stroke, thickness, rect_fill, id3);
    add_regagon(x, y-100, n, r, stroke, thickness, reg_fill, id4);
    add_triangle(x, y+100, b, h, stroke, thickness, tri_fill, id5);

    add_canvas(cw, ch, cx, cy);
    return 0;
}


**************************************************
**************************************************
bugsyc
**************************************************

#!/bin/bash
```

```
filebase=$(echo $1|cut -f1 -d. )
./bugsy.native $1 > "$filebase.ll"
llc "$filebase.ll" -o "$filebase.s"
g++ "$filebase.s" "_build/builtins.o" -no-pie -lglut -lGL -lGLU -lGLEW
    -o $filebase
rm "$filebase.ll" "$filebase.s"

echo "$filebase compiled"


**************************************************
testall.sh
**************************************************

#!/bin/bash

RED='\033[0;31m'
GREEN='\033[0;32m'
YELLOW='\033[0;33m'
NC='\033[0m' # No Color

function red {
   printf "${RED}$@${NC}\n"
}

function green {
   printf "${GREEN}$@${NC}\n"
}

function yellow {
   printf "${YELLOW}$@${NC}\n"
}

# Regression testing script for bugsy
# 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 bugsy compiler. Usually "./bugsy.native"
```

```
# Try "_build/bugsy.native" if ocamlbuild was unable to create a
    symbolic link.
BUGSY="./bugsy.native"
#BUGSY="_build/microc.native"

# Set time limit for all operations
ulimit -t 30

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

keep=0

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

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

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

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

```
    }
}

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

Check() {
    error=0
    basename=‘echo $1 | sed ’s/.*\\///
                          s/.bug//’‘
    reffile=‘echo $1 | sed ’s/.bug$//’ | sed ’s/test\/tests//’‘
    basedir="‘echo $1 | sed ’s/\/[^\/]*$//’‘../golden_set"
    echo -n "$basename..."

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

    if [ ! -d tmp ]; then
        mkdir tmp
    fi
    generatedfiles=""

    generatedfiles="$generatedfiles tmp/${basename}.ll tmp/${basename}.s
        tmp/${basename}.exe tmp/${basename}.out" &&
    Run "$BUGSY" "$1" ">" "tmp/${basename}.ll" &&
    Run "$LLC" "-relocation-model=pic" "tmp/${basename}.ll" ">"
        "tmp/${basename}.s" &&
    #Run "$CC" "-o" "${basename}.exe" "${basename}.s" "printbig.o" &&
    Run "g++" "tmp/${basename}.s" "_build/builtins.o" "-no-pie" "-lglut"
        "-lGL" "-lGLU" "-lGLEW" "-o" "tmp/${basename}.exe" &&
    Run "DEBUG=1 ./tmp/${basename}.exe" > "tmp/${basename}.out" &&
    Compare "test/golden_set/${basename}.out" "tmp/${reffile}.out"
        "tmp/${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 $(green "OK")
    #echo "Hi"
```

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

CheckFail() {
    error=0
    basename=‘echo $1 | sed ’s/.*\\///
                          s/.bug//’‘
    reffile=‘echo $1 | sed ’s/.bug$//’ | sed ’s/test\/tests//’‘
    basedir="‘echo $1 | sed ’s/\/[^\/]*$//’‘../fails"

    echo -n "$basename..."

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

    generatedfiles=""

    generatedfiles="$generatedfiles tmp/${basename}.err
        tmp/${basename}.diff" &&
    RunFail "$BUGSY" "<" $1 "2>" "tmp/${basename}.err" ">>" "$globallog"
        &&
    Compare "test/fails/${basename}.err" "tmp/${basename}.err"
        "tmp/${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 $(green "OK")
   echo "###### SUCCESS" 1>&2
    else
   echo "###### FAILED" 1>&2
   globalerror=$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 _build/builtins.o ]
then
    echo "Could not find builtins.o"
    echo "Have you run \"make\"?"
    exit 1
fi

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

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

exit $globalerror

\subsection{Tests}
```

```
**************************************************
**************************************************
fail-array.bug
**************************************************

num main() {
    num i;
    string s;
    num[2] x;
    x = [1,2];
    s = "hello";
    x[0] = s;
    return 0.0;
}

**************************************************
fail-assign1.bug
**************************************************

num main()
{
  num i;
  bool b;

  i = 42;
  i = 10;
  b = true;
  b = false;
  i = false; /* Fail: assigning a bool to a num */
}


**************************************************
fail-assign2.bug
**************************************************

num main()
{
  num i;
  bool b;

  b = 48; /* Fail: assigning an integer to a bool */
}


**************************************************
fail-assign3.bug
**************************************************
```

```
void myvoid()
{
  return;
}

num main()
{
  num i;

  i = myvoid(); /* Fail: assigning a void to an integer */
}
```

```
**************************************************
fail-dead1.bug
**************************************************

num main()
{
  num i;

  i = 15;
  return i;
  i = 32; /* Error: code after a return */
}
```

```
**************************************************
fail-dead2.bug
**************************************************

num main()
{
  num i;

  {
    i = 15;
    return i;
  }
  i = 32; /* Error: code after a return */
}
```

```
**************************************************
fail-expr1.bug
**************************************************

num a;
bool b;
```

```
void foo(num c, bool d)
{
  num dd;
  bool e;
  a + c;
  c - a;
  a * 3;
  c / 2;
  d + a; /* Error: bool + num */
}

num main()
{
  return 0;
}
```

```
**************************************************
fail-expr2.bug
**************************************************

num a;
bool b;

void foo(num c, bool d)
{
  num d;
  bool e;
  b + a; /* Error: bool + num */
}

num main()
{
  return 0;
}
```

```
**************************************************
fail-for1.bug
**************************************************

num main()
{

  num j;
  for (i = 0 ; i < 10 ; i = i + 1) {
    j = j + 1;
  }
```

```
  return 0;
}


**************************************************
fail-for2.bug
**************************************************

num main()
{
  num i;

  for(i = 0; j < 10 ; i = i + 1) {} /* j undefined */

  return 0;
}


**************************************************
fail-for3.bug
**************************************************

num main()
{
  num i;

  for(i = 0; i ; i = i + 1) {} /* i is an num, not Boolean */

  return 0;
}


**************************************************
fail-for4.bug
**************************************************

num main()
{
  num i;

  for(i = 0; i < 10 ; i = j + 1) {} /* j undefined */

  return 0;
}


**************************************************
fail-for5.bug
**************************************************
```

```
num main()
{
  num i;

  for(i = 0; i < 10 ; i = i + 1) {
    foo(); /* Error: no function foo */
  }

  return 0;
}
```

```
**************************************************
fail-func1.bug
**************************************************

num foo() {}

num bar() {}

num baz() {}

void bar() {} /* Error: duplicate function bar */

num main()
{
  return 0;
}
```

```
**************************************************
fail-func2.bug
**************************************************

num foo(num a, bool b, num c) { }

void bar(num a, bool b, num a) {} /* Error: duplicate formal a in bar */

num main()
{
  return 0;
}
```

```
**************************************************
fail-func3.bug
**************************************************

num foo(num a, bool b, num c) { }
```

```
void bar(num a, void b, num c) {} /* Error: illegal void formal b */

num main()
{
  return 0;
}
```

```
**************************************************
fail-func4.bug
**************************************************

num foo() {}

void bar() {}

num print() {} /* Should not be able to define print */

void baz() {}

num main()
{
  return 0;
}
```

```
**************************************************
fail-func5.bug
**************************************************

num foo() {}

num bar() {
  num a;
  void b; /* Error: illegal void local b */
  bool c;

  return 0;
}

num main()
{
  return 0;
}
```

```
**************************************************
fail-func6.bug
**************************************************
```

```
void foo(num a, bool b)
{
}

num main()
{
  foo(42, true);
  foo(42); /* Wrong number of arguments */
}


**************************************************
fail-func7.bug
**************************************************

void foo(num a, bool b)
{
}

num main()
{
  foo(42, true);
  foo(42, true, false); /* Wrong number of arguments */
}


**************************************************
fail-func8.bug
**************************************************

void foo(num a, bool b)
{
}

void bar()
{
}

num main()
{
  foo(42, true);
  foo(42, bar()); /* num and void, not num and bool */
}


**************************************************
fail-func9.bug
**************************************************

void foo(num a, bool b)
```

```
{
}

num main()
{
  foo(42, true);
  foo(42, 42); /* Fail: num, not bool */
}


**************************************************
fail-global1.bug
**************************************************

num c;
bool b;
void a; /* global variables should not be void */


num main()
{
  return 0;
}


**************************************************
fail-global2.bug
**************************************************

num b;
bool c;
num a;
num b; /* Duplicate global variable */

num main()
{
  return 0;
}


**************************************************
fail-if1.bug
**************************************************

num main()
{
  if(true) {}
  if(false) {} else {}
  if(42) {} /* Error: non-bool predicate */
}
```

```
**************************************************
fail-if2.bug
**************************************************

num main()
{
  if(true) {
    foo; /* Error: undeclared variable */
  }
}


**************************************************
fail-if3.bug
**************************************************

num main()
{
  if(true) {
    42;
  } else {
    bar; /* Error: undeclared variable */
  }
}


**************************************************
fail-nomain.bug
**************************************************



**************************************************
fail-return1.bug
**************************************************

num main()
{
  return true; /* Should return num */
}


**************************************************
fail-return2.bug
**************************************************

void foo()
{
```

```
  if(true) {
    return 42; /* Should return void */
  } else {
    return;
  }
}

num main()
{
  return 42;
}
```

```
**************************************************
fail-while1.bug
**************************************************

num main()
{
  num i;

  while(true) {
    i = i + 1;
  }

  while(42) { /* Should be bool */
    i = i + 1;
  }

}
```

```
**************************************************
fail-while2.bug
**************************************************

num main()
{
  num i;

  while (true) {
    i = i + 1;
  }

  while(true) {
    foo(); /* foo undefined */
  }

}
```

```
**************************************************
test-add1.bug
**************************************************

num m;
num n;

num add(num x, num y)
{
  return x + y;
}

num main()
{
  m = 17.1;
  n = 25.0;
  print( add(m, n) );
  return 0;
}


**************************************************
test-arith1.bug
**************************************************

num x;
num y;
num z;
num main()
{
  x = 39.0;
  y = 3.0;
  z = x + y;
  print(z);
  return 0.0;
}


**************************************************
test-arith2.bug
**************************************************

num main()
{
  print(1 + 2 * 3 + 4);
  return 0;
}
```

```
**************************************************
test-arith3.bug
**************************************************

num foo(num a)
{
  return a;
}

num main()
{
  num a;
  a = 42;
  a = a + 5;
  print(a);
  return 0;
}


**************************************************
test-array1.bug
**************************************************

num main() {
    num i;
    num[5] x;
    x = [1.0, 2.0, 3.0, 4.0, 5.0];
    for (i = 0; i < 5; i = i + 1) {
        print(x[i]);
    }
    return 0.0;
}


**************************************************
test-array2.bug
**************************************************

num main() {
    num[2] arr;
    num x;
    x = 5.0;
    arr = [1.0, 2.0];
    print(arr[0]);
    arr[0] = x;
    print(arr[0]);
    return 0;
}
```

```
**************************************************
test-circle.bug
**************************************************

num main() {
    num x;
    num y;
    num r;
    num h;
    num w;
    num cx;
    num cy;
    string stroke;
    string fill;
    num thickness;
    string id;
    id = "";
    fill = "0.2 0.2 0.2";
    stroke = "0.0 0.3 0.5";
    cx = 0;
    cy = 0;
    h = 1000;
    w = 1500;
    x = h / 2;
    y = h / 2;
    r = 100;
    thickness = 5;
    init_canvas();
    id = add_circle(x, y, r, stroke, thickness, fill, id);
    add_canvas(w, h, cx, cy);
    return 0;
}


**************************************************
test-decrement1.bug
**************************************************

num main() {
    num a;
    a = 5;
    print(a);
    print(a--);
    print(a);
    return 0;
}

**************************************************
test-decrement2.bug
**************************************************
```

```
num main() {
    num b;
    b = 2;
    print(b);
    print(--b);
    print(b);
    return 0;
}
```

****
test-ellipse.bug
****

```
num main() {
    num cx;
    num cy;
    num scale;
    num x;
    num y;
    num cw;
    num ch;
    num r;
    num w;
    num h;
    num s;
    num n;
    num b;
    cx = 0;
    cy = 0;
    cw = 1500;
    scale = 3;
    ch = 1000;
    x = cw / 2;
    y = ch / 2;
    r = 100;
    s = 75;
    n = 10;
    b = 60;
    w = 100;
    h = 100;

    string stroke;
    string reg_fill;
    string tri_fill;
    string rect_fill;
    string sq_fill;
    string cir_fill;
    string ell_fill;
```

```
    stroke = "0.0 0.3 0.5";
    reg_fill = "0.2 0.2 0.2";
    cir_fill = "1.0 0.2 0.6";
    ell_fill = "0.8 0.5 0.5";
    sq_fill = "0.9 0.9 0.2";
    tri_fill = "0.7 0.4 0.0";
    rect_fill = "1.0 0.9 0.6";

    num thickness;
    thickness = 5;
    string id;
    id = "";


    init_canvas();
    add_ellipse(x+300, y, w, h*2, stroke, thickness, ell_fill, id);
    add_canvas(cw, ch, cx, cy);

    return 0;
}
```

```
**************************************************
test-fib.bug
**************************************************

num fib(num x)
{
  if(x < 2) {
    return 1;
  }
  return fib(x-1) + fib(x-2);
}

num main()
{
  print(fib(0));
  print(fib(1));
  print(fib(2));
  print(fib(3));
  print(fib(4));
  print(fib(5));
  return 0;
}


**************************************************
test-for1.bug
**************************************************

num main()
```

```
{
  num i;
  for (i = 0 ; i < 5 ; i = i + 1) {
    print(i);
  }
  print(42);
  return 0;
}
```

```
**************************************************
test-for2.bug
**************************************************
```

```
num main()
{
  num i;
  i = 0.0;
  for ( ; i < 5; ) {
    print(i);
    i = i + 1;
  }
  print(42);
  return 0;
}
```

```
**************************************************
test-func1.bug
**************************************************
```

```
num add(num a, num b)
{
  return a + b;
}

num main()
{
  num a;
  a = add(39, 3);
  print(a);
  return 0;
}
```

```
**************************************************
test-func2.bug
**************************************************
```

```
num fun(num x, num y)
```

```
{
  return 0;
}

num main()
{
  num i;
  i = 1;

  fun(i = 2.0, i = i+1);

  print(i);
  return 0;
}
```

```
**************************************************
test-func3.bug
**************************************************

void printem(num a, num b, num c, num d)
{
  print(a);
  print(b);
  print(c);
  print(d);
}

num main()
{
  printem(42,17,192,8);
  return 0;
}
```

```
**************************************************
test-func4.bug
**************************************************

num add(num a, num b)
{
  num c;
  c = a + b;
  return c;
}

num main()
{
  num d;
  d = add(52, 10);
```

```
  print(d);
  return 0;
}


**************************************************
test-func5.bug
**************************************************

num foo(num a)
{
  return a;
}

num main()
{
  return 0;
}


**************************************************
test-func6.bug
**************************************************

void foo() {}

num bar(num a, bool b, num c) { return a + c; }

num main()
{
  print(bar(17, false, 25));
  return 0;
}


**************************************************
test-func7.bug
**************************************************

num a;

void foo(num c)
{
  a = c + 42;
}

num main()
{
  foo(73);
  print(a);
```

```
  return 0;
}


**************************************************
test-func8.bug
**************************************************

void foo(num a)
{
  print(a + 3);
}

num main()
{
  foo(40);
  return 0;
}


**************************************************
test-gcd.bug
**************************************************

num gcd(num a, num b) {
  while (a != b) {
    if (a > b) {
      a = a - b;
    }
    else {
      b = b - a;
    }
  }
  return a;
}

num main()
{
  print(gcd(2,14));
  print(gcd(3,15));
  print(gcd(99,121));
  return 0;
}


**************************************************
test-gcd2.bug
**************************************************

num gcd(num a, num b) {
```

```
  while (a != b) {
    if (a > b) {
      a = a - b;
    }
    else {
      b = b - a;
    }
  }
  return a;
}

num main()
{
  print(gcd(14,21));
  print(gcd(8,36));
  print(gcd(99,121));
  return 0;
}


**************************************************
test-global1.bug
**************************************************

num a;
num b;

void printa()
{
  print(a);
}

void printbee()
{
  print(b);
}

void incab()
{
  a = a + 1;
  b = b + 1;
}

num main()
{
  a = 42;
  b = 21;
  printa();
  printbee();
  incab();
```

```
  printa();
  printbee();
  return 0.0;
}


**************************************************
test-global2.bug
**************************************************

bool i;

num main()
{
  num i; /* Should hide the global i */

  i = 42;
  print(i + i);
  return 0;
}


**************************************************
test-global3.bug
**************************************************

num i;
bool b;
num j;

num main()
{
  i = 42;
  j = 10;
  print(i + j);
  return 0;
}


**************************************************
test-hello.bug
**************************************************

num main()
{
  print(42);
  print(71);
  print(1);
  return 0;
}
```

```
**************************************************
test-if1.bug
**************************************************

num main()
{
  if (true) {
    print(42);
  }
  print(17);
  return 0;
}


**************************************************
test-if2.bug
**************************************************

num main()
{
  if (true) {
    print(42);
  }
  else {
    print(8);
  }
  print(17);
  return 0;
}


**************************************************
test-if3.bug
**************************************************

num main()
{
  if (false) {
    print(42);
  }
  print(17);
  return 0;
}


**************************************************
test-if4.bug
**************************************************
```

```
num main()
{
  if (false) {
    print(42);
  }
  else {
    print(8);
  }
  print(17);
  return 0;
}
```

```
**************************************************
test-if5.bug
**************************************************

num cond(bool b)
{
  num x;
  if (b) {
    x = 42;
  }

  else {
    x = 17;
  }
  return x;
}

num main()
{
 print(cond(true));
 print(cond(false));
 return 0;
}
```

```
**************************************************
test-increment1.bug
**************************************************

num main() {
    num a;
    a = 5;
    print(a);
    print(a++);
    print(a);
    return 0;
```

```
}

**************************************************
test-increment2.bug
**************************************************

num main() {
    num b;
    b = 2;
    print(b);
    print(++b);
    print(b);
    return 0;
}

**************************************************
test-local1.bug
**************************************************

void foo(bool i)
{
  num i; /* Should hide the formal i */

  i = 42;
  print(i + i);
}

num main()
{
  foo(true);
  return 0;
}


**************************************************
test-local2.bug
**************************************************

num foo(num a, bool b)
{
  num c;
  bool d;

  c = a;

  return c + 10;
}

num main() {
 print(foo(37, false));
```

```
 return 0;
}


**************************************************
test-moveBy.bug
**************************************************

num main() {
    num cx;
    num cy;
    num scale;
    num x;
    num y;
    num cw;
    num ch;
    num r;
    num w;
    num h;
    num s;
    num n;
    num b;
    cx = 0;
    cy = 0;
    cw = 1500;
    scale = 3;
    ch = 1000;
    x = cw / 2;
    y = ch / 2;
    r = 100;
    s = 75;
    n = 10;
    b = 60;
    w = 100;
    h = 100;

    string stroke;
    string reg_fill;
    string tri_fill;
    string rect_fill;
    string sq_fill;
    string cir_fill;
    string ell_fill;

    stroke = "0.0 0.3 0.5";
    reg_fill = "0.2 0.2 0.2";
    cir_fill = "1.0 0.2 0.6";
    ell_fill = "0.8 0.5 0.5";
    sq_fill = "0.9 0.9 0.2";
    tri_fill = "0.7 0.4 0.0";
```

```
    rect_fill = "1.0 0.9 0.6";

    num thickness;
    thickness = 5;
    string id;
    id = "";

    init_canvas();
    id = add_regagon(x, y-100, n, r, stroke, thickness, reg_fill, id);
    moveById(id, 500, 500, 1);
    add_canvas(cw, ch, cx, cy);

    return 0;
}


**************************************************
test-ops1.bug
**************************************************

num main()
{
  print(1 + 2);
  print(1 - 2);
  print(1 * 2);
  print(100 / 2);
  print(99);
  printb(1 =? 2);
  printb(1 =? 1);
  print(99);
  printb(1 != 2);
  printb(1 != 1);
  print(99);
  printb(1 < 2);
  printb(2 < 1);
  print(99);
  printb(1 <= 2);
  printb(1 <= 1);
  printb(2 <= 1);
  print(99);
  printb(1 > 2);
  printb(2 > 1);
  print(99);
  printb(1 >= 2);
  printb(1 >= 1);
  printb(2 >= 1);
  return 0;
}
```

```
**************************************************
test-ops2.bug
**************************************************

num main()
{
  num i;
  i=43;
  printb(true);
  printb(false);
  printb(true and true);
  printb(true and false);
  printb(false and true);
  printb(false and false);
  printb(true or true);
  printb(true or false);
  printb(false or true);
  printb(false or false);
  printb(!false);
  printb(!true);
  print(-10);
  print(--i);
}


**************************************************
test-rectangle.bug
**************************************************

num main() {
    num cx;
    num cy;
    num scale;
    num x;
    num y;
    num cw;
    num ch;
    num r;
    num w;
    num h;
    num s;
    num n;
    num b;
    cx = 0;
    cy = 0;
    cw = 1500;
    scale = 3;
    ch = 1000;
    x = cw / 2;
    y = ch / 2;
```

```
    r = 100;
    s = 75;
    n = 10;
    b = 60;
    w = 100;
    h = 100;

    string stroke;
    string reg_fill;
    string tri_fill;
    string rect_fill;
    string sq_fill;
    string cir_fill;
    string ell_fill;

    stroke = "0.0 0.3 0.5";
    reg_fill = "0.2 0.2 0.2";
    cir_fill = "1.0 0.2 0.6";
    ell_fill = "0.8 0.5 0.5";
    sq_fill = "0.9 0.9 0.2";
    tri_fill = "0.7 0.4 0.0";
    rect_fill = "1.0 0.9 0.6";

    num thickness;
    thickness = 5;
    string id;
    id = "";


    init_canvas();
    add_rectangle(x, y-300, w*2, h, stroke, thickness, rect_fill, id);
    add_canvas(cw, ch, cx, cy);
    return 0;
}

**************************************************
test-regagon.bug
**************************************************

num main() {
    num cx;
    num cy;
    num scale;
    num x;
    num y;
    num cw;
    num ch;
    num r;
    num w;
    num h;
```

```
    num s;
    num n;
    num b;
    cx = 0;
    cy = 0;
    cw = 1500;
    scale = 3;
    ch = 1000;
    x = cw / 2;
    y = ch / 2;
    r = 100;
    s = 75;
    n = 10;
    b = 60;
    w = 100;
    h = 100;

    string stroke;
    string reg_fill;
    string tri_fill;
    string rect_fill;
    string sq_fill;
    string cir_fill;
    string ell_fill;

    stroke = "0.0 0.3 0.5";
    reg_fill = "0.2 0.2 0.2";
    cir_fill = "1.0 0.2 0.6";
    ell_fill = "0.8 0.5 0.5";
    sq_fill = "0.9 0.9 0.2";
    tri_fill = "0.7 0.4 0.0";
    rect_fill = "1.0 0.9 0.6";

    num thickness;
    thickness = 5;
    string id;
    id = "";

    init_canvas();
    add_regagon(x, y-100, n, r, stroke, thickness, reg_fill, id);
    add_canvas(cw, ch, cx, cy);

    return 0;
}

**************************************************
test-rotateBy.bug
**************************************************

num main() {
```

```
num cx;
num cy;
num scale;
num x;
num y;
num cw;
num ch;
num r;
num w;
num h;
num s;
num n;
num b;
cx = 0;
cy = 0;
cw = 1500;
scale = 3;
ch = 1000;
x = cw / 2;
y = ch / 2;
r = 100;
s = 75;
n = 10;
b = 60;
w = 100;
h = 100;

string stroke;
string reg_fill;
string tri_fill;
string rect_fill;
string sq_fill;
string cir_fill;
string ell_fill;

stroke = "0.0 0.3 0.5";
reg_fill = "0.2 0.2 0.2";
cir_fill = "1.0 0.2 0.6";
ell_fill = "0.8 0.5 0.5";
sq_fill = "0.9 0.9 0.2";
tri_fill = "0.7 0.4 0.0";
rect_fill = "1.0 0.9 0.6";

num thickness;
thickness = 5;
string id;
id = "";

init_canvas();
// id = add_regagon(x, y-100, n, r, stroke, thickness, reg_fill, id);
```

```
    id = add_triangle(x, y, b, h, stroke, thickness, tri_fill, id);
    rotateById(id, 360, 1);
    add_canvas(cw, ch, cx, cy);

    return 0;
}


**************************************************
test-scaleBy.bug
**************************************************

num main() {
    num cx;
    num cy;
    num scale;
    num x;
    num y;
    num cw;
    num ch;
    num r;
    num w;
    num h;
    num s;
    num n;
    num b;
    cx = 0;
    cy = 0;
    cw = 1500;
    scale = 3;
    ch = 1000;
    x = cw / 2;
    y = ch / 2;
    r = 100;
    s = 75;
    n = 10;
    b = 60;
    w = 100;
    h = 100;

    string stroke;
    string reg_fill;
    string tri_fill;
    string rect_fill;
    string sq_fill;
    string cir_fill;
    string ell_fill;

    stroke = "0.0 0.3 0.5";
    reg_fill = "0.2 0.2 0.2";
```

```
    cir_fill = "1.0 0.2 0.6";
    ell_fill = "0.8 0.5 0.5";
    sq_fill = "0.9 0.9 0.2";
    tri_fill = "0.7 0.4 0.0";
    rect_fill = "1.0 0.9 0.6";

    num thickness;
    thickness = 5;
    string id;
    id = "";

    init_canvas();
    id = add_regagon(x, y-100, n, r, stroke, thickness, reg_fill, id);
    scaleById(id, 3, 1);
    add_canvas(cw, ch, cx, cy);

    return 0;
}


**************************************************
test-square.bug
**************************************************

num main() {
    num cx;
    num cy;
    num scale;
    num x;
    num y;
    num cw;
    num ch;
    num r;
    num w;
    num h;
    num s;
    num n;
    num b;
    cx = 0;
    cy = 0;
    cw = 1500;
    scale = 3;
    ch = 1000;
    x = cw / 2;
    y = ch / 2;
    r = 100;
    s = 75;
    n = 10;
    b = 60;
    w = 100;
```

```
    h = 100;

    string stroke;
    string reg_fill;
    string tri_fill;
    string rect_fill;
    string sq_fill;
    string cir_fill;
    string ell_fill;

    stroke = "0.0 0.3 0.5";
    sq_fill = "0.9 0.9 0.2";

    num thickness;
    thickness = 5;
    string id;
    id = "";


    init_canvas();
    add_square(x, y+300, s, stroke, thickness, sq_fill, id);
    add_canvas(cw, ch, cx, cy);
    return 0;
}

**************************************************
test-triangle.bug
**************************************************

num main() {
    num cx;
    num cy;
    num scale;
    num x;
    num y;
    num cw;
    num ch;
    num r;
    num w;
    num h;
    num s;
    num n;
    num b;
    cx = 0;
    cy = 0;
    cw = 1500;
    scale = 3;
    ch = 1000;
    x = cw / 2;
    y = ch / 2;
```

```
    r = 100;
    s = 75;
    n = 10;
    b = 60;
    w = 100;
    h = 100;

    string stroke;
    string reg_fill;
    string tri_fill;
    string rect_fill;
    string sq_fill;
    string cir_fill;
    string ell_fill;

    stroke = "0.0 0.3 0.5";
    reg_fill = "0.2 0.2 0.2";
    cir_fill = "1.0 0.2 0.6";
    ell_fill = "0.8 0.5 0.5";
    sq_fill = "0.9 0.9 0.2";
    tri_fill = "0.7 0.4 0.0";
    rect_fill = "1.0 0.9 0.6";

    num thickness;
    thickness = 5;
    string id;
    id = "";

    init_canvas();
    add_triangle(x, y+100, b, h, stroke, thickness, tri_fill, id);
    add_canvas(cw, ch, cx, cy);
    return 0;
}
```

```
**************************************************
test-var1.bug
**************************************************

num main()
{
  num a;
  a = 42;
  print(a);
  return 0;
}


**************************************************
test-var2.bug
**************************************************
```

```
num a;

void foo(num c)
{
  a = c + 42;
}

num main()
{
  foo(73);
  print(a);
  return 0;
}
```

```
**************************************************
test-while1.bug
**************************************************

num main()
{
  num i;
  i = 5;
  while (i > 0) {
    print(i);
    i = i - 1;
  }
  print(42);
  return 0;
}
```

```
**************************************************
test-while2.bug
**************************************************

num foo(num a)
{
  num j;
  j = 0;
  while (a > 0) {
    j = j + 2;
    a = a - 1;
  }
  return j;
}

num main()
{
```

```
  print(foo(7));
  return 0;
}
```

****

8.2 Examples

CODE:
Sample Program 1
arrays.bug

```
num main() {

   num[5] x;
   num a;
   num b;
   num c;
   num d;
   num e;
   x = [10.9, 30.6, 10.3, 5.4, 6.7];

   printf("The array should be [10.9, 30.6, 10.3, 5.4, 6.7]");
   printf("The first element is");
   print(x[0]);


   a = 2.1;

   b = 2;

   e = x[b];

   e = 4.0;

   x[2] = a;

   c = 3.4;
   x[3] = c;


   x[4] = 9.0;

   printf("sirs second element is");
   printf("sirs third element is");
   print(x[3]);
   printf("sirs fourth element is");
   print(x[4]);
```

```
}


LLVM for program 1:
; ModuleID = 'Bugsy'
source_filename = "Bugsy"

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@fmt.1 = private unnamed_addr constant [4 x i8] c"%g\0A\00", align 1
@fmt.2 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@str = private unnamed_addr constant [49 x i8] c"The array should be
    [10.9, 30.6, 10.3, 5.4, 6.7]\00", align 1
@str.3 = private unnamed_addr constant [21 x i8] c"The first element
    is\00", align 1
@str.4 = private unnamed_addr constant [23 x i8] c"sirs second element
    is\00", align 1
@str.5 = private unnamed_addr constant [22 x i8] c"sirs third element
    is\00", align 1
@str.6 = private unnamed_addr constant [23 x i8] c"sirs fourth element
    is\00", align 1

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

declare double @demo()

declare double @add_point_xy(double, double, i8*)

declare i8* @add_circle(double, double, double, i8*, double, i8*, i8*)

declare i8* @add_triangle(double, double, double, double, i8*, double,
    i8*, i8*)

declare i8* @add_square(double, double, double, i8*, double, i8*, i8*)

declare i8* @add_rectangle(double, double, double, double, i8*, double,
    i8*, i8*)

declare i8* @add_regagon(double, double, double, double, i8*, double,
    i8*, i8*)

declare i8* @add_ellipse(double, double, double, double, i8*, double,
    i8*, i8*)

declare double @add_canvas(double, double, double, double)

declare double @moveById(i8*, double, double, double)

declare double @scaleById(i8*, double, double)
```

```
declare double @rotateById(i8*, double, double)

declare double @init_canvas()

define i32 @main() {
entry:
  %x = alloca [5 x double]
  %a = alloca double
  %b = alloca double
  %c = alloca double
  %d = alloca double
  %e = alloca double
  store [5 x double] [double 1.090000e+01, double 3.060000e+01, double
      1.030000e+01, double 5.400000e+00, double 6.700000e+00], [5 x
      double]* %x
  %printf = call double (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.2, i32 0, i32 0), i8* getelementptr
      inbounds ([49 x i8], [49 x i8]* @str, i32 0, i32 0))
  %printf1 = call double (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.2, i32 0, i32 0), i8* getelementptr
      inbounds ([21 x i8], [21 x i8]* @str.3, i32 0, i32 0))
  %x2 = getelementptr inbounds [5 x double], [5 x double]* %x, i32 0,
      i32 0
  %x3 = load double, double* %x2
  %printf4 = call double (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.1, i32 0, i32 0), double %x3)
  store double 2.100000e+00, double* %a
  store double 2.000000e+00, double* %b
  %b5 = load double, double* %b
  %trunc = fptosi double %b5 to i32
  %x6 = getelementptr inbounds [5 x double], [5 x double]* %x, i32 0,
      i32 %trunc
  %x7 = load double, double* %x6
  store double %x7, double* %e
  store double 4.000000e+00, double* %e
  %x8 = getelementptr [5 x double], [5 x double]* %x, i32 0, i32 3
  %a9 = load double, double* %a
  %trunc10 = fptosi double %a9 to i32
  store double %a9, double* %x8
  store double 3.400000e+00, double* %c
  %x11 = getelementptr [5 x double], [5 x double]* %x, i32 0, i32 3
  %c12 = load double, double* %c
  %trunc13 = fptosi double %c12 to i32
  store double %c12, double* %x11
  %x14 = getelementptr [5 x double], [5 x double]* %x, i32 0, i32 3
  store double 9.000000e+00, double* %x14
  %printf15 = call double (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.2, i32 0, i32 0), i8* getelementptr
      inbounds ([23 x i8], [23 x i8]* @str.4, i32 0, i32 0))
```

```
  %printf16 = call double (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.2, i32 0, i32 0), i8* getelementptr
      inbounds ([22 x i8], [22 x i8]* @str.5, i32 0, i32 0))
  %x17 = getelementptr inbounds [5 x double], [5 x double]* %x, i32 0,
      i32 3
  %x18 = load double, double* %x17
  %printf19 = call double (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.1, i32 0, i32 0), double %x18)
  %printf20 = call double (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.2, i32 0, i32 0), i8* getelementptr
      inbounds ([23 x i8], [23 x i8]* @str.6, i32 0, i32 0))
  %x21 = getelementptr inbounds [5 x double], [5 x double]* %x, i32 0,
      i32 4
  %x22 = load double, double* %x21
  %printf23 = call double (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.1, i32 0, i32 0), double %x22)
  ret i32 0
}
```

Sample Program 2:
shape_test.bug

```
num main() {
    num cx;
    num cy;
    num scale;
    num x;
    num y;
    num cw;
    num ch;
    num r;
    num w;
    num h;
    num s;
    num n;
    num b;
    cx = 0;
    cy = 0;
    cw = 1500;
    scale = 3;
    ch = 1000;
    x = cw / 2;
    y = ch / 2;
    r = 100;
    s = 75;
    n = 10;
    b = 60;
    w = 100;
```

```
    h = 100;

    string stroke;
    string reg_fill;
    string tri_fill;
    string rect_fill;
    string sq_fill;
    string cir_fill;
    string ell_fill;

    stroke = "0.0 0.3 0.5";
    reg_fill = "0.2 0.2 0.2";
    cir_fill = "1.0 0.2 0.6";
    ell_fill = "0.8 0.5 0.5";
    sq_fill = "0.9 0.9 0.2";
    tri_fill = "0.7 0.4 0.0";
    rect_fill = "1.0 0.9 0.6";

    num thickness;
    thickness = 5;
    string id;
    string id2;
    string id3;
    string id4;
    string id5;
    string id6;
    id = "";
    id2 = "";
    id3 = "";
    id4 = "";
    id5 = "";
    id6 = "";


    init_canvas();

    add_ellipse(x+300, y, w, h*2, stroke, thickness, ell_fill, id6);
    add_circle(x-300, y, r, stroke, thickness, cir_fill, id);
    add_square(x, y+300, s, stroke, thickness, sq_fill, id2);
    add_rectangle(x, y-300, w*2, h, stroke, thickness, rect_fill, id3);
    add_regagon(x, y-100, n, r, stroke, thickness, reg_fill, id4);
    add_triangle(x, y+100, b, h, stroke, thickness, tri_fill, id5);

    add_canvas(cw, ch, cx, cy);
    return 0;
}

The LLVM for program 2:

; ModuleID = 'Bugsy'
```

```
source_filename = "Bugsy"

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@fmt.1 = private unnamed_addr constant [4 x i8] c"%g\0A\00", align 1
@fmt.2 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@str = private unnamed_addr constant [12 x i8] c"0.0 0.3 0.5\00", align 1
@str.3 = private unnamed_addr constant [12 x i8] c"0.2 0.2 0.2\00",
    align 1
@str.4 = private unnamed_addr constant [12 x i8] c"1.0 0.2 0.6\00",
    align 1
@str.5 = private unnamed_addr constant [12 x i8] c"0.8 0.5 0.5\00",
    align 1
@str.6 = private unnamed_addr constant [12 x i8] c"0.9 0.9 0.2\00",
    align 1
@str.7 = private unnamed_addr constant [12 x i8] c"0.7 0.4 0.0\00",
    align 1
@str.8 = private unnamed_addr constant [12 x i8] c"1.0 0.9 0.6\00",
    align 1
@str.9 = private unnamed_addr constant [1 x i8] zeroinitializer, align 1
@str.10 = private unnamed_addr constant [1 x i8] zeroinitializer, align 1
@str.11 = private unnamed_addr constant [1 x i8] zeroinitializer, align 1
@str.12 = private unnamed_addr constant [1 x i8] zeroinitializer, align 1
@str.13 = private unnamed_addr constant [1 x i8] zeroinitializer, align 1
@str.14 = private unnamed_addr constant [1 x i8] zeroinitializer, align 1

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

declare double @demo()

declare double @add_point_xy(double, double, i8*)

declare i8* @add_circle(double, double, double, i8*, double, i8*, i8*)

declare i8* @add_triangle(double, double, double, double, i8*, double,
    i8*, i8*)

declare i8* @add_square(double, double, double, i8*, double, i8*, i8*)

declare i8* @add_rectangle(double, double, double, double, i8*, double,
    i8*, i8*)

declare i8* @add_regagon(double, double, double, double, i8*, double,
    i8*, i8*)

declare i8* @add_ellipse(double, double, double, double, i8*, double,
    i8*, i8*)

declare double @add_canvas(double, double, double, double)

declare double @moveById(i8*, double, double, double)
```

```
declare double @scaleById(i8*, double, double)

declare double @rotateById(i8*, double, double)

declare double @init_canvas()

define i32 @main() {
entry:
  %cx = alloca double
  %cy = alloca double
  %scale = alloca double
  %x = alloca double
  %y = alloca double
  %cw = alloca double
  %ch = alloca double
  %r = alloca double
  %w = alloca double
  %h = alloca double
  %s = alloca double
  %n = alloca double
  %b = alloca double
  %stroke = alloca i8*
  %reg_fill = alloca i8*
  %tri_fill = alloca i8*
  %rect_fill = alloca i8*
  %sq_fill = alloca i8*
  %cir_fill = alloca i8*
  %ell_fill = alloca i8*
  %thickness = alloca double
  %id = alloca i8*
  %id2 = alloca i8*
  %id3 = alloca i8*
  %id4 = alloca i8*
  %id5 = alloca i8*
  %id6 = alloca i8*
  store double 0.000000e+00, double* %cx
  store double 0.000000e+00, double* %cy
  store double 1.500000e+03, double* %cw
  store double 3.000000e+00, double* %scale
  store double 1.000000e+03, double* %ch
  %cw1 = load double, double* %cw
  %tmp = fdiv double %cw1, 2.000000e+00
  store double %tmp, double* %x
  %ch2 = load double, double* %ch
  %tmp3 = fdiv double %ch2, 2.000000e+00
  store double %tmp3, double* %y
  store double 1.000000e+02, double* %r
  store double 7.500000e+01, double* %s
  store double 1.000000e+01, double* %n
```

```
store double 6.000000e+01, double* %b
store double 1.000000e+02, double* %w
store double 1.000000e+02, double* %h
store i8* getelementptr inbounds ([12 x i8], [12 x i8]* @str, i32 0,
    i32 0), i8** %stroke
store i8* getelementptr inbounds ([12 x i8], [12 x i8]* @str.3, i32 0,
    i32 0), i8** %reg_fill
store i8* getelementptr inbounds ([12 x i8], [12 x i8]* @str.4, i32 0,
    i32 0), i8** %cir_fill
store i8* getelementptr inbounds ([12 x i8], [12 x i8]* @str.5, i32 0,
    i32 0), i8** %ell_fill
store i8* getelementptr inbounds ([12 x i8], [12 x i8]* @str.6, i32 0,
    i32 0), i8** %sq_fill
store i8* getelementptr inbounds ([12 x i8], [12 x i8]* @str.7, i32 0,
    i32 0), i8** %tri_fill
store i8* getelementptr inbounds ([12 x i8], [12 x i8]* @str.8, i32 0,
    i32 0), i8** %rect_fill
store double 5.000000e+00, double* %thickness
store i8* getelementptr inbounds ([1 x i8], [1 x i8]* @str.9, i32 0,
    i32 0), i8** %id
store i8* getelementptr inbounds ([1 x i8], [1 x i8]* @str.10, i32 0,
    i32 0), i8** %id2
store i8* getelementptr inbounds ([1 x i8], [1 x i8]* @str.11, i32 0,
    i32 0), i8** %id3
store i8* getelementptr inbounds ([1 x i8], [1 x i8]* @str.12, i32 0,
    i32 0), i8** %id4
store i8* getelementptr inbounds ([1 x i8], [1 x i8]* @str.13, i32 0,
    i32 0), i8** %id5
store i8* getelementptr inbounds ([1 x i8], [1 x i8]* @str.14, i32 0,
    i32 0), i8** %id6
%init_canvas = call double @init_canvas()
%id64 = load i8*, i8** %id6
%ell_fill5 = load i8*, i8** %ell_fill
%thickness6 = load double, double* %thickness
%stroke7 = load i8*, i8** %stroke
%h8 = load double, double* %h
%tmp9 = fmul double %h8, 2.000000e+00
%w10 = load double, double* %w
%y11 = load double, double* %y
%x12 = load double, double* %x
%tmp13 = fadd double %x12, 3.000000e+02
%add_ellipse = call i8* @add_ellipse(double %tmp13, double %y11,
    double %w10, double %tmp9, i8* %stroke7, double %thickness6, i8*
    %ell_fill5, i8* %id64)
%id14 = load i8*, i8** %id
%cir_fill15 = load i8*, i8** %cir_fill
%thickness16 = load double, double* %thickness
%stroke17 = load i8*, i8** %stroke
%r18 = load double, double* %r
%y19 = load double, double* %y
```

```
%x20 = load double, double* %x
%tmp21 = fsub double %x20, 3.000000e+02
%add_circle = call i8* @add_circle(double %tmp21, double %y19, double
    %r18, i8* %stroke17, double %thickness16, i8* %cir_fill15, i8*
    %id14)
%id222 = load i8*, i8** %id2
%sq_fill23 = load i8*, i8** %sq_fill
%thickness24 = load double, double* %thickness
%stroke25 = load i8*, i8** %stroke
%s26 = load double, double* %s
%y27 = load double, double* %y
%tmp28 = fadd double %y27, 3.000000e+02
%x29 = load double, double* %x
%add_square = call i8* @add_square(double %x29, double %tmp28, double
    %s26, i8* %stroke25, double %thickness24, i8* %sq_fill23, i8*
    %id222)
%id330 = load i8*, i8** %id3
%rect_fill31 = load i8*, i8** %rect_fill
%thickness32 = load double, double* %thickness
%stroke33 = load i8*, i8** %stroke
%h34 = load double, double* %h
%w35 = load double, double* %w
%tmp36 = fmul double %w35, 2.000000e+00
%y37 = load double, double* %y
%tmp38 = fsub double %y37, 3.000000e+02
%x39 = load double, double* %x
%add_rectangle = call i8* @add_rectangle(double %x39, double %tmp38,
    double %tmp36, double %h34, i8* %stroke33, double %thickness32,
    i8* %rect_fill31, i8* %id330)
%id440 = load i8*, i8** %id4
%reg_fill41 = load i8*, i8** %reg_fill
%thickness42 = load double, double* %thickness
%stroke43 = load i8*, i8** %stroke
%r44 = load double, double* %r
%n45 = load double, double* %n
%y46 = load double, double* %y
%tmp47 = fsub double %y46, 1.000000e+02
%x48 = load double, double* %x
%add_regagon = call i8* @add_regagon(double %x48, double %tmp47,
    double %n45, double %r44, i8* %stroke43, double %thickness42, i8*
    %reg_fill41, i8* %id440)
%id549 = load i8*, i8** %id5
%tri_fill50 = load i8*, i8** %tri_fill
%thickness51 = load double, double* %thickness
%stroke52 = load i8*, i8** %stroke
%h53 = load double, double* %h
%b54 = load double, double* %b
%y55 = load double, double* %y
%tmp56 = fadd double %y55, 1.000000e+02
%x57 = load double, double* %x
```

```
  %add_triangle = call i8* @add_triangle(double %x57, double %tmp56,
      double %b54, double %h53, i8* %stroke52, double %thickness51, i8*
      %tri_fill50, i8* %id549)
  %cy58 = load double, double* %cy
  %cx59 = load double, double* %cx
  %ch60 = load double, double* %ch
  %cw61 = load double, double* %cw
  %add_canvas = call double @add_canvas(double %cw61, double %ch60,
      double %cx59, double %cy58)
  ret i32 0
}
```

Example Program 3:
animation-example-array.bug

```
num main() {
    num cx;
    num cy;
    num scale;
    num x;
    num y;
    num cw;
    num ch;
    num r;
    num w;
    num h;
    num s;
    num n;
    num b;
    cx = 0;
    cy = 0;
    cw = 1500;
    ch = 1000;
    x = cw / 2;
    y = ch / 2;


    string stroke;
    string cir_fill;

    stroke = "0.0 0.3 0.5";
    cir_fill = "1.0 0.2 0.6";

    string id;
    id = "";

    init_canvas();

    num i;
    for(i = 0; i < 10; i++) {
```

```
        id = add_circle(x+2000, y+2000, 50 + (i * 10), stroke, 5,
            cir_fill, id);
        moveById(id, -2000, -2000, 1);
    }

    add_canvas(cw, ch, cx, cy);
}
```

LLVM for program 3:

```
; ModuleID = 'Bugsy'
source_filename = "Bugsy"

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@fmt.1 = private unnamed_addr constant [4 x i8] c"%g\0A\00", align 1
@fmt.2 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@str = private unnamed_addr constant [12 x i8] c"0.0 0.3 0.5\00", align 1
@str.3 = private unnamed_addr constant [12 x i8] c"1.0 0.2 0.6\00",
    align 1
@str.4 = private unnamed_addr constant [1 x i8] zeroinitializer, align 1

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

declare double @demo()

declare double @add_point_xy(double, double, i8*)

declare i8* @add_circle(double, double, double, i8*, double, i8*, i8*)

declare i8* @add_triangle(double, double, double, double, i8*, double,
    i8*, i8*)

declare i8* @add_square(double, double, double, i8*, double, i8*, i8*)

declare i8* @add_rectangle(double, double, double, double, i8*, double,
    i8*, i8*)

declare i8* @add_regagon(double, double, double, double, i8*, double,
    i8*, i8*)

declare i8* @add_ellipse(double, double, double, double, i8*, double,
    i8*, i8*)

declare double @add_canvas(double, double, double, double)

declare double @moveById(i8*, double, double, double)

declare double @scaleById(i8*, double, double)

declare double @rotateById(i8*, double, double)
```

```
declare double @init_canvas()

define i32 @main() {
entry:
  %cx = alloca double
  %cy = alloca double
  %scale = alloca double
  %x = alloca double
  %y = alloca double
  %cw = alloca double
  %ch = alloca double
  %r = alloca double
  %w = alloca double
  %h = alloca double
  %s = alloca double
  %n = alloca double
  %b = alloca double
  %stroke = alloca i8*
  %cir_fill = alloca i8*
  %id = alloca i8*
  %i = alloca double
  store double 0.000000e+00, double* %cx
  store double 0.000000e+00, double* %cy
  store double 1.500000e+03, double* %cw
  store double 1.000000e+03, double* %ch
  %cw1 = load double, double* %cw
  %tmp = fdiv double %cw1, 2.000000e+00
  store double %tmp, double* %x
  %ch2 = load double, double* %ch
  %tmp3 = fdiv double %ch2, 2.000000e+00
  store double %tmp3, double* %y
  store i8* getelementptr inbounds ([12 x i8], [12 x i8]* @str, i32 0,
      i32 0), i8** %stroke
  store i8* getelementptr inbounds ([12 x i8], [12 x i8]* @str.3, i32 0,
      i32 0), i8** %cir_fill
  store i8* getelementptr inbounds ([1 x i8], [1 x i8]* @str.4, i32 0,
      i32 0), i8** %id
  %init_canvas = call double @init_canvas()
  store double 0.000000e+00, double* %i
  br label %while

while:                                            ; preds = %while_body,
    %entry
  %i18 = load double, double* %i
  %tmp19 = fcmp olt double %i18, 1.000000e+01
  br i1 %tmp19, label %while_body, label %merge

while_body:                                       ; preds = %while
  %id4 = load i8*, i8** %id
```

```
  %cir_fill5 = load i8*, i8** %cir_fill
  %stroke6 = load i8*, i8** %stroke
  %i7 = load double, double* %i
  %tmp8 = fmul double %i7, 1.000000e+01
  %tmp9 = fadd double 5.000000e+01, %tmp8
  %y10 = load double, double* %y
  %tmp11 = fadd double %y10, 2.000000e+03
  %x12 = load double, double* %x
  %tmp13 = fadd double %x12, 2.000000e+03
  %add_circle = call i8* @add_circle(double %tmp13, double %tmp11,
      double %tmp9, i8* %stroke6, double 5.000000e+00, i8* %cir_fill5,
      i8* %id4)
  store i8* %add_circle, i8** %id
  %id14 = load i8*, i8** %id
  %moveById = call double @moveById(i8* %id14, double -2.000000e+03,
      double -2.000000e+03, double 1.000000e+00)
  %i15 = load double, double* %i
  %tmp16 = fadd double %i15, 1.000000e+00
  %tmp17 = fsub double %i15, 1.000000e+00
  store double %tmp16, double* %i
  br label %while

merge:                                            ; preds = %while
  %cy20 = load double, double* %cy
  %cx21 = load double, double* %cx
  %ch22 = load double, double* %ch
  %cw23 = load double, double* %cw
  %add_canvas = call double @add_canvas(double %cw23, double %ch22,
      double %cx21, double %cy20)
  ret i32 0
}
```

8.3 Git logs

```
GIT LOG:
commit 4e7972cb9a70cafa75c832a5da819b21a7857c1a
Merge: 1ce200d 7632be8
Author: mwinitch <michaelwin26@gmail.com>
Date:   Tue Apr 27 01:08:55 2021 +0000

    Merge branch 'master' of https://github.com/BenjaminSnyder/bugsy

commit 1ce200d98078bca432c6f50e106a761e5f070980
Author: mwinitch <michaelwin26@gmail.com>
Date:   Tue Apr 27 01:08:44 2021 +0000

    Cleaned up file
```

```
commit 7632be8d6a432b1a467de57821673d3a4b3d682a
Author: Ben <bs3148@columbia.edu>
Date:  Tue Apr 27 00:48:59 2021 +0000

    removed executable

commit 1ed1411c638b8cbe09fe0981a8412c77adc525d3
Merge: 82ded78 53e13a5
Author: Ben <bs3148@columbia.edu>
Date:  Tue Apr 27 00:47:53 2021 +0000

    Merge branch 'master' of github.com:BenjaminSnyder/bugsy

commit 82ded78dd84b90b18e1dba6fb8e6cd9c9a3e79fe
Author: Ben <bs3148@columbia.edu>
Date:  Tue Apr 27 00:47:49 2021 +0000

    removed old presentation file

commit 53e13a55f6360454d9695d8a259febbf828193a9
Merge: e1ea7a9 9e29684
Author: Jason <jasoncardinale19@gmail.com>
Date:  Mon Apr 26 20:41:57 2021 -0400

    Merge branch 'master' of github.com:BenjaminSnyder/bugsy

commit e1ea7a90b13dd98d3978de6c6165c1543431b240
Author: Jason <jasoncardinale19@gmail.com>
Date:  Mon Apr 26 20:41:50 2021 -0400

    animation array test

commit 9e296849a276e3bf49171ad08be8d8be02e49e07
Author: Ben <bs3148@columbia.edu>
Date:  Tue Apr 27 00:37:29 2021 +0000

    updated compile script to be bugsyc

commit 2946880ab2e6aa9f037207cc220f7f3f98c7b76d
Author: Ben <bs3148@columbia.edu>
Date:  Tue Apr 27 00:28:16 2021 +0000

    removed unnecessary files and updated gitignore and made bugsyc our
    compile script

commit ccc9bfb0c1f4ca2c1dea5d8281f1cd51df722efa
Merge: da36eee 618d7f2
Author: evantilley <elt2141@columbia.edu>
Date:  Tue Apr 27 00:25:54 2021 +0000
```

```
    Merge branch 'master' of github.com:BenjaminSnyder/bugsy

commit da36eee79a052a1eb1f4840e38956212b693a891
Author: evantilley <elt2141@columbia.edu>
Date:  Tue Apr 27 00:25:39 2021 +0000

    removed dump module output

commit 618d7f2e023d5f8e158651dc648b62e4a73d094b
Author: Ben <bs3148@columbia.edu>
Date:  Tue Apr 27 00:22:27 2021 +0000

    cleaned examples folder

commit b9cd79e76b105990b66f4fe245d1402d34289f8e
Merge: 9f6492b 9d84c58
Author: evantilley <elt2141@columbia.edu>
Date:  Tue Apr 27 00:16:56 2021 +0000

    Merge branch 'master' of github.com:BenjaminSnyder/bugsy

commit 9f6492bdb584f4330cc275277f90b539f93cc85c
Author: evantilley <elt2141@columbia.edu>
Date:  Tue Apr 27 00:16:46 2021 +0000

    more code clean up

commit 9d84c589d2f793d02bbcd964cc52662c9ee8cd5d
Author: mwinitch <michaelwin26@gmail.com>
Date:  Tue Apr 27 00:16:10 2021 +0000

    removed comment

commit 1328e6ae135ae9271c9d120dd9049cb6f6baab99
Merge: c6dc4a4 b81c671
Author: Jason <jasoncardinale19@gmail.com>
Date:  Mon Apr 26 20:13:41 2021 -0400

    Merge branch 'master' of github.com:BenjaminSnyder/bugsy

commit c6dc4a493fc97688d70ff65ebe26f0948f37599b
Author: Jason <jasoncardinale19@gmail.com>
Date:  Mon Apr 26 20:13:36 2021 -0400

    shape_test back

commit dc6fc62dc5a19b4ef5a631b94823d7f2f06ae721
Merge: a547c2d aab9dd9
Author: evantilley <elt2141@columbia.edu>
Date:  Mon Apr 26 23:49:27 2021 +0000
```

```
    cleaning up

commit b81c671161253696aeecf26e6e96eb1395dc2294
Author: Benjamin Snyder <bs3148@columbia.edu>
Date:  Mon Apr 26 19:46:14 2021 -0400

    Update README

commit 5093212bd000501b700e06484578032f36160c61
Merge: 2f9c580 763f5cf
Author: Jason <jasoncardinale19@gmail.com>
Date:  Mon Apr 26 19:45:11 2021 -0400

    Merge branch 'master' of github.com:BenjaminSnyder/bugsy

commit 2f9c580c88bc1b1f6601700a227841e0d6bdd63c
Author: Jason <jasoncardinale19@gmail.com>
Date:  Mon Apr 26 19:45:05 2021 -0400

    builtins cleanup

commit 763f5cff1ef4ec53f83a86815f4e761977a76d7b
Author: Benjamin Snyder <bs3148@columbia.edu>
Date:  Mon Apr 26 19:40:05 2021 -0400

    Update .gitignore

commit 187edf5d09b8b5c601a15d5384260a9787f30784
Author: Benjamin Snyder <bs3148@columbia.edu>
Date:  Mon Apr 26 19:39:30 2021 -0400

    Update _tags

commit d850ef3a20436fa882af659f5166a99c6523c1cc
Author: Benjamin Snyder <bs3148@columbia.edu>
Date:  Mon Apr 26 19:39:03 2021 -0400

    Update README

commit 587970263a83967914c3e978d162ed08684327df
Author: Jason <jasoncardinale19@gmail.com>
Date:  Mon Apr 26 19:30:04 2021 -0400

    semant cleanup

commit a547c2d594445d7bf0cfcb0711789e71cd73455c
Author: Jason <jasoncardinale19@gmail.com>
Date:  Mon Apr 26 19:24:51 2021 -0400
```

```
    array err

commit aab9dd9d45c405df5cf528693d8255fc088ab142
Author: evantilley <elt2141@columbia.edu>
Date:   Mon Apr 26 23:24:39 2021 +0000

    allowed variable array assignments + added example

commit f3d1b73756bad4578b501b21bf52826825a3237b
Author: mwinitch <michaelwin26@gmail.com>
Date:   Mon Apr 26 23:11:15 2021 +0000

    Updating test files in the Makefile

commit 5fadef816c13ef6b1612270ca121a9a8dbfd7063
Author: Jason <jasoncardinale19@gmail.com>
Date:   Mon Apr 26 18:57:44 2021 -0400

    fixed rotate and array test

commit 304ad9a291d5d2200b6356e6208f7fac7f599404
Author: mwinitch <michaelwin26@gmail.com>
Date:   Mon Apr 26 22:50:08 2021 +0000

    Array fail test

commit 88dc91384fd28b7b2cb66009cdfdb1f90d99aab5
Author: Ben <bs3148@columbia.edu>
Date:   Mon Apr 26 22:48:05 2021 +0000

    fix makefile

commit 0ea510361240ceb0f98c7cc2b0d1bcbb045e0110
Merge: c9c6472 82e1dc1
Author: Jason <jasoncardinale19@gmail.com>
Date:   Mon Apr 26 18:35:04 2021 -0400

    Merge branch 'master' of github.com:BenjaminSnyder/bugsy

commit c9c647214d371665257f893b9ecea4b86f7d573b
Author: Jason <jasoncardinale19@gmail.com>
Date:   Mon Apr 26 18:34:45 2021 -0400

    fixed rotate timing

commit 82e1dc102ce12c7fab30f21bcdb3b6137eae67b0
Author: mwinitch <michaelwin26@gmail.com>
Date:   Mon Apr 26 21:44:22 2021 +0000

    Comments for parser
```

```
commit c286dd649a4bcc9a012789899197ca36e727bac0
Author: mwinitch <michaelwin26@gmail.com>
Date:  Mon Apr 26 21:04:56 2021 +0000

    Cleaned testing folder

commit 110ea168adf48d0e927d286650c41ba809cb091a
Author: Jason <jasoncardinale19@gmail.com>
Date:  Mon Apr 26 16:56:32 2021 -0400

    Commit message

commit 39a579a878ad4fcaa45f7dfca34773bea9a9d361
Author: mwinitch <michaelwin26@gmail.com>
Date:  Mon Apr 26 20:38:46 2021 +0000

    Added increment and decrement testing

commit f9cee5471680b9cdb9a4ed31129711deee7cd331
Author: Jason <jasoncardinale19@gmail.com>
Date:  Mon Apr 26 16:21:29 2021 -0400

    fixed the golden set

commit a3ce2382602621b89a88c4d1408284260f9f8709
Author: Jason <jasoncardinale19@gmail.com>
Date:  Mon Apr 26 15:54:45 2021 -0400

    memset shapes to 0

commit 0b2ed0239967701685500 36f7f950df96b5d2e6a
Author: Ben <bs3148@columbia.edu>
Date:  Mon Apr 26 01:33:28 2021 +0000

    fixed all warnings

commit 2b2285216dacea7d1ba1201d382def6d7ebfe443
Author: mwinitch <michaelwin26@gmail.com>
Date:  Mon Apr 26 01:10:17 2021 +0000

    Taking care of warnings in codegen

commit 50c2309bb159663684c83fa762da46f73b8b5a2e
Merge: 54528a1 ee5d6cf
Author: mwinitch <michaelwin26@gmail.com>
Date:  Mon Apr 26 00:36:09 2021 +0000

    Merge branch 'master' into add_test_shape
```

```
commit ee5d6cf80102e308e423f9580b1d6379a6692470
Author: Ben <bs3148@columbia.edu>
Date:  Mon Apr 26 00:35:09 2021 +0000

    fixed tests

commit 54528a1be42fafce00afabb6745fba5526d0facb
Author: mwinitch <michaelwin26@gmail.com>
Date:  Mon Apr 26 00:33:29 2021 +0000

    Shape tests

commit 65a79ad18750ee23c43e87b2a3975f70756576fa
Author: Ben <bs3148@columbia.edu>
Date:  Mon Apr 26 00:26:31 2021 +0000

    fixed function calls

commit 40ddf136065d768f6ce143cb112226fa1630bcd0
Author: evantilley <elt2141@columbia.edu>
Date:  Mon Apr 26 00:15:35 2021 +0000

    working on last touches

commit 04a04afdab109b4284ce520bbd427f609faddbf9
Author: Ben <bs3148@columbia.edu>
Date:  Mon Apr 26 00:08:09 2021 +0000

    removed extraneous files

commit 0aeafaa885789bc2ae21ebc2e174b3cca321ee2f
Merge: 5c5f35a 785af7c
Author: Ben <bs3148@columbia.edu>
Date:  Mon Apr 26 00:07:39 2021 +0000

    Merge branch 'master' of github.com:BenjaminSnyder/bugsy

commit 5c5f35a177db6f2dab5dc38a028101f2af7d5a0d
Author: Ben <bs3148@columbia.edu>
Date:  Mon Apr 26 00:07:30 2021 +0000

    cleaned files

commit fcb4facbe197d365b33839c742102cdf632e90e4
Author: Ben <bs3148@columbia.edu>
Date:  Mon Apr 26 00:04:02 2021 +0000

    fixed pattern matching warnings

commit 785af7cc3da628420e941c5f2de9486f5c35296d
```

```
Merge: babd93e baad79d
Author: Jason <jasoncardinale19@gmail.com>
Date:  Sun Apr 25 20:01:12 2021 -0400

    Merge branch 'master' into classes

commit babd93e3db24b2ce83e7124368750638f39fc2a3
Author: Jason <jasoncardinale19@gmail.com>
Date:  Sun Apr 25 19:55:44 2021 -0400

    presentation demo program

commit baad79dbdea8fb4eba0562f581332f03df095df0
Author: mwinitch <michaelwin26@gmail.com>
Date:  Sun Apr 25 23:51:02 2021 +0000

    Added circle testing

commit ea4aa3e815c160cbfc361f5ab7c06427787f7e0c
Author: Jason <jasoncardinale19@gmail.com>
Date:  Sun Apr 25 19:43:00 2021 -0400

    shapes all goods

commit d90f01892481d52c4f55920bf31b7cdd5617c5c1
Merge: dd18c09 3ee36cd
Author: mwinitch <michaelwin26@gmail.com>
Date:  Sun Apr 25 22:21:09 2021 +0000

    Fixed testing

commit 3ee36cda3548a281d4f81e7486a94b8db98b0417
Author: evantilley <elt2141@columbia.edu>
Date:  Sun Apr 25 21:49:37 2021 +0000

    fixed up tests

commit 03153a48062c8b2c83ca3bc97805efed9f5ac9c7
Merge: 58481a0 9e4b75d
Author: evantilley <elt2141@columbia.edu>
Date:  Sun Apr 25 21:27:58 2021 +0000

    merge

commit dd18c09e112a479ce7bd5cefa153393bca3c5f09
Author: Jason <jasoncardinale19@gmail.com>
Date:  Sun Apr 25 17:26:39 2021 -0400

    shapetest
```

```
commit b7a41da3028aee79d463451697c644237466d36f
Author: Jason <jasoncardinale19@gmail.com>
Date:  Sun Apr 25 17:14:05 2021 -0400

    shape cleanup stuff

commit ce9e84807658edb617758c27f9cb7fa4d92c0baf
Author: mwinitch <michaelwin26@gmail.com>
Date:  Sun Apr 25 20:39:59 2021 +0000

    Added shapes to codegen

commit 58481a0d956ef8885c973106932144e93d0e3d70
Author: evantilley <elt2141@columbia.edu>
Date:  Sun Apr 25 20:32:20 2021 +0000

    arrays are workinggit add .git add .

commit 9c1adfa9dad6c1a287baa294fd21eff4358ea40d
Author: Jason <jasoncardinale19@gmail.com>
Date:  Sun Apr 25 16:14:17 2021 -0400

    animations working for multiple objects and back to back animations

commit 0c506ef2be0297a71b205662199e6949070e087b
Author: mwinitch <michaelwin26@gmail.com>
Date:  Sun Apr 25 18:15:30 2021 +0000

    Added scaling to Bugsy shapes

commit 387499fa896bd1820b4c25cbceef6cfd334e34b0
Author: Jason <jasoncardinale19@gmail.com>
Date:  Sun Apr 25 13:56:27 2021 -0400

    bugsy circle and moveby is working woot woot

commit 2f70efca2759010f1b66f1e5a37a071b3975d37d
Author: Jason <jasoncardinale19@gmail.com>
Date:  Sun Apr 25 11:36:47 2021 -0400

    trying to get shape tests to work

commit b9ca5788e83a4584631d827f3de58db69beb1575
Author: evantilley <elt2141@columbia.edu>
Date:  Sun Apr 25 11:46:11 2021 +0000

    close

commit f8d68ba7012049efd3dd3a50ab739d7eff509c8f
Author: evantilley <elt2141@columbia.edu>
```

```
Date:   Sun Apr 25 10:17:41 2021 +0000

    so close to working arrays

commit 9e4b75dc60264e978274a16f7ee40d2505849ea7
Author: Ben <bs3148@columbia.edu>
Date:   Sun Apr 25 00:16:54 2021 +0000

    fixed struct ptr for classes

commit a1228365190a682c1112f5fd6ecac08fe3d4d3c8
Author: evantilley <elt2141@columbia.edu>
Date:   Sun Apr 25 00:16:53 2021 +0000

    working on arrays sooo close

commit a144937713e6759734697e83cc2f3b4c931a528e
Merge: 37828f8 810ca06
Author: mwinitch <michaelwin26@gmail.com>
Date:   Sat Apr 24 23:52:05 2021 +0000

    Merge branch 'classes' of https://github.com/BenjaminSnyder/bugsy
        into classes

commit 37828f8a92cdfa9276c02349bdd22ba523d474fe
Author: mwinitch <michaelwin26@gmail.com>
Date:   Sat Apr 24 23:51:55 2021 +0000

    codegen updates and tests

commit 810ca0651f74c046c7f25e2b2e998d5c4fad8229
Author: Jason <jasoncardinale19@gmail.com>
Date:   Sat Apr 24 19:12:18 2021 -0400

    fixed builtin shape names

commit 043310fc07b93def9605c8c5cd0168472e3f41fc
Author: evantilley <elt2141@columbia.edu>
Date:   Sat Apr 24 04:38:35 2021 +0000

    interesting point at arrays

commit b57a3237b1141b32650fcf25e35f6eb3569dc50c
Merge: 46f913c ba54bac
Author: Sofia <sofia.sanchez@columbia.edu>
Date:   Sat Apr 24 00:30:39 2021 -0400

    Merge branch 'classes' of github.com:BenjaminSnyder/bugsy into
        classes
```

```
commit 46f913c925f0377587eb8e72f68781585bde1657
Author: Sofia <sofia.sanchez@columbia.edu>
Date:  Sat Apr 24 00:30:22 2021 -0400

    stdlib shape classes ftw

commit ba54bac1f807e2946a590b1beff51b447dbef5b2
Author: Ben <bs3148@columbia.edu>
Date:  Fri Apr 23 21:54:43 2021 +0000

    Added instancevariables to classTyp after semantic checking

commit 15ab3897f7d0ddeb0f4524f0c67743a569e5d7bb
Author: mwinitch <michaelwin26@gmail.com>
Date:  Thu Apr 22 01:40:12 2021 +0000

    Added testing for fail cases

commit fb161091aab77cc4e59c7e62bb3836a48d89f5f2
Author: mwinitch <michaelwin26@gmail.com>
Date:  Wed Apr 21 23:52:24 2021 +0000

    Adding semant checking for accessing class variables

commit f0be3e7d618d9da686c3cfc44d37eec348406741
Author: evantilley <elt2141@columbia.edu>
Date:  Mon Apr 19 01:02:58 2021 +0000

    array accessing now working as expected!

commit 62815ce72168418e1154cef58a29045342718738
Author: evantilley <elt2141@columbia.edu>
Date:  Mon Apr 19 00:11:53 2021 +0000

    array assigning now works

commit 5be2a850687fa6355d9967e3b454f77f6a7903cf
Author: evantilley <elt2141@columbia.edu>
Date:  Sun Apr 18 21:48:17 2021 +0000

    semant now working for arrays sirs

commit a7eb1836485fc6203bfbf54c6fc71e2e234f2494
Author: Ben <bs3148@columbia.edu>
Date:  Sat Apr 17 23:50:07 2021 +0000

    class access is BROKE

commit a4d2a26249548c89b43a6451804db98c2569ea1d
Author: Ben <bs3148@columbia.edu>
```

```
Date:   Sat Apr 17 23:35:02 2021 +0000

    merged

commit b948b7069bd3d9ccc4de39dcbe3d8310257f9e95
Author: mwinitch <michaelwin26@gmail.com>
Date:   Sat Apr 17 23:33:46 2021 +0000

    Fixed semant to recognize class variables in class function

commit 8c3d8b7d03c3fbc566ba4ff76c5dc0e3e3bc41ee
Author: mwinitch <michaelwin26@gmail.com>
Date:   Sat Apr 17 22:29:50 2021 +0000

    Adding semant checking for class functions

commit 8e447d99637cf8d70de05d69510780ce785b3795
Merge: 1ee6564 c0126bd
Author: Ben <bs3148@columbia.edu>
Date:   Sat Apr 17 19:21:55 2021 +0000

    fixed merge issues

commit c0126bd963ce2f4b6b7eb0d6422bc832558a90e9
Merge: 6b07cf3 613580c
Author: Ben <bs3148@columbia.edu>
Date:   Sat Apr 17 19:07:48 2021 +0000

    fixed merge conflict

commit 1ee6564146152b0df550a611370ad29b8b63f913
Merge: 6b07cf3 613580c
Author: Ben <bs3148@columbia.edu>
Date:   Sat Apr 17 19:07:48 2021 +0000

    fixed merge conflict

commit 6b07cf31dbb7e4978923446061508d2553af8048
Author: Ben <bs3148@columbia.edu>
Date:   Sat Apr 17 19:04:59 2021 +0000

    fixed constructor returns and assigning a class

commit 613580cd8b058e602f720af5c8b58cb50d5e08fc
Author: Ben <bs3148@columbia.edu>
Date:   Sat Apr 17 19:04:59 2021 +0000

    fixed constructor returns and assigning a class

commit 4da546f4be10c86593b5073722778966031b4f2e
```

```
Author: Ben <bs3148@columbia.edu>
Date:  Tue Apr 13 07:39:41 2021 +0000

    updated sast and added some semant changes

commit d15b6d3df49e340551f71c935a27342af44fba3e
Author: evantilley <elt2141@columbia.edu>
Date:  Fri Apr 9 08:18:45 2021 +0000

    arrays mildly working, stay tuned...

commit b6515fccd99a531fde4baf1bafb50742c5c176e3
Author: Ben <bs3148@columbia.edu>
Date:  Fri Apr 9 02:58:14 2021 +0000

    fixed classes and constructor in ast

commit 434d8013d9f9f119c41d2347027c6653924d1569
Author: mwinitch <michaelwin26@gmail.com>
Date:  Fri Apr 9 02:21:59 2021 +0000

    Chaned the ast

commit cc71a1900517ddef0aeb65acf2b2fdafe06e3f35
Author: Ben <bs3148@columbia.edu>
Date:  Fri Apr 9 01:53:45 2021 +0000

    fixed ast printing

commit f3eedcfca098ea7d9130b5075edfa9f9dd670f67
Author: mwinitch <michaelwin26@gmail.com>
Date:  Fri Apr 9 01:38:40 2021 +0000

    Added code to instantiate an object from class

commit 11f582a6ea4a011a16b33c5815cad15c4b71421e
Author: Ben <bs3148@columbia.edu>
Date:  Fri Apr 9 00:43:33 2021 +0000

    removed deprecated tests

commit 63a5e139e07df243896abef55412a9cf31cfa1c5
Merge: cd4175a 6820ca4
Author: Ben <bs3148@columbia.edu>
Date:  Thu Apr 8 07:20:13 2021 +0000

    Merge branch 'master' of github.com:BenjaminSnyder/bugsy

commit cd4175a9f0b9fadc382cda27463f9bf8e5256113
Author: Ben <bs3148@columbia.edu>
```

```
Date:   Thu Apr 8 07:15:06 2021 +0000

    did some spring cleaning, new directories and updated test script
        and Makefile to work with new dirs

commit 6820ca40f076a2b0f46b18ef528fcd051041165e
Author: Ben <bs3148@columbia.edu>
Date:   Thu Apr 8 07:15:06 2021 +0000

    did some spring cleaning, new directories and updated test script
        and Makefile to work with new dirs

commit 1b05e437a773efd32d607dd7ac45ff95ab5eb269
Author: evantilley <elt2141@columbia.edu>
Date:   Thu Apr 8 07:00:34 2021 +0000

    arrays finally compiling again after glitch in matrices

commit 818d5b1aa3ef30bfb106605f438b9610c28db7b6
Author: Ben <bs3148@columbia.edu>
Date:   Wed Apr 7 07:06:30 2021 +0000

    fixed boolean printing and now all tests pass

commit cd7d9880bdd8daa3d6a004bc40c8c1a6d14548e8
Author: Ben <bs3148@columbia.edu>
Date:   Wed Apr 7 06:57:33 2021 +0000

    added rudimentary support for increment/decrement in semant and
        codegen, and removed whitespace from files

commit f46429a2a62de6628b6a6ddc17b4384e78f9fafa
Author: mwinitch <michaelwin26@gmail.com>
Date:   Wed Apr 7 05:21:39 2021 +0000

    Updated testing

commit a2a63e8bab49a60f616daee2ca6827970e1a462b
Merge: 13980ba 961c73c
Author: mwinitch <michaelwin26@gmail.com>
Date:   Wed Apr 7 04:47:57 2021 +0000

    Merge remote-tracking branch 'origin/master' into add_tests

commit 13980ba970223e6bc343da0651b47cdb12860195
Author: mwinitch <michaelwin26@gmail.com>
Date:   Wed Apr 7 04:46:43 2021 +0000

    Update test
```

```
commit 961c73c0c271540e8d859a702ab5420fe0213305
Author: evantilley <elt2141@columbia.edu>
Date:  Wed Apr 7 04:45:39 2021 +0000

    merging fixed return type to master

commit 80d99ab4e5921c960ebc66d5518cf0d97cd7c3d8
Author: evantilley <elt2141@columbia.edu>
Date:  Wed Apr 7 04:43:50 2021 +0000

    fixed return type of main by modifying codegen

commit 2fa3595545e95a7829d716a0823b39db83cfb7ae
Author: evantilley <elt2141@columbia.edu>
Date:  Wed Apr 7 03:46:17 2021 +0000

    possibly fixed return 0

commit dcdb20ea199e240269720d0b408411790035908c
Merge: 3d89031 c5b68fb
Author: mwinitch <michaelwin26@gmail.com>
Date:  Wed Apr 7 01:08:42 2021 +0000

    Merging

commit c5b68fb9c299b428ccf32c0d1741db53b705499d
Author: evantilley <elt2141@columbia.edu>
Date:  Sun Apr 4 08:18:02 2021 +0000

    arrays compile now! (and also semant checking seems reasonable),
        check out arrays.bug

commit 8b07fab1c5a78136bc45c058f0d63ada6eeb8706
Merge: ff6fd54 236a2e7
Author: evantilley <elt2141@columbia.edu>
Date:  Sun Apr 4 05:52:15 2021 +0000

    Merge branch 'arrays' of github.com:BenjaminSnyder/bugsy into arrays

commit ff6fd541081a4ba016e7a91ca289de09f074ff88
Author: evantilley <elt2141@columbia.edu>
Date:  Sun Apr 4 05:52:11 2021 +0000

    some codegen stuff but nothing too important, just committing so I
        can pull

commit 236a2e7d5f61a3eb465b8ebe57d6ca607e9e9a60
Author: mwinitch <michaelwin26@gmail.com>
Date:  Sun Apr 4 04:57:48 2021 +0000
```

```
    Updating semant

commit 3d89031984d54610819ee5ede1e7d8f00b00ffa6
Author: mwinitch <michaelwin26@gmail.com>
Date:   Sun Apr 4 01:07:07 2021 +0000

    More test fixing

commit 51c54cef31ebb046ec79d2c4e1752f02b474022c
Author: Jason <jasoncardinale19@gmail.com>
Date:   Sat Apr 3 21:03:26 2021 -0400

    add_square and fixed canvas scaling issue

commit a31e57eb89fedd3aaad59954a52c45365c21a4b8
Author: sofia <ss5664@columbia.edu>
Date:   Sat Apr 3 22:38:09 2021 +0000

    added squayure

commit 9fa318508450b284d20cabd506da59795a650146
Author: Jason <jasoncardinale19@gmail.com>
Date:   Sat Apr 3 18:05:18 2021 -0400

    added canvas

commit a092319625fcb699c4b8f9e91f9e2643a2c40b04
Author: Jason <jasoncardinale19@gmail.com>
Date:   Thu Apr 1 16:31:01 2021 -0400

    Added a builtin function for drawing a circle

commit 023b3f48e62a2335cc58d7cc49ac552d7102a023
Author: Ben <bs3148@columbia.edu>
Date:   Tue Mar 30 19:46:40 2021 +0000

    added array literal,decls, and access

commit 9c661b33bd7005b50d7c456959a33c5cdbc8b0dc
Author: Ben <bs3148@columbia.edu>
Date:   Tue Mar 30 08:01:08 2021 +0000

    added a testclass for pretty print tabbing

commit a9502e0ce537d57583c21929503271ff231a4407
Author: Ben <bs3148@columbia.edu>
Date:   Tue Mar 30 07:50:02 2021 +0000

    added shapes and arrays and dot notation to scanner, parser, ast
```

```
commit ddc77d28d54d53f5a9e3f583ebf8e0b0cffd0cee
Author: Ben <bs3148@columbia.edu>
Date:  Tue Mar 30 07:25:05 2021 +0000

    HOLY PRINTING

commit 3f3b7d9b11aea3f5e6f1052607127755bcc9ede8
Author: Ben <bs3148@columbia.edu>
Date:  Tue Mar 30 07:11:58 2021 +0000

    added shapes and refactored scanner and parser and ast, added tabs
        to pretty printer

commit 1027d2aa619db1143b087620609f7ca888785d78
Author: Ben <bs3148@columbia.edu>
Date:  Sun Mar 28 22:38:57 2021 +0000

    changed gitignore and updated demo compile script

commit 429e9ceeda5db10d59aa667e4922431fa35fd889
Author: Ben <bs3148@columbia.edu>
Date:  Wed Mar 24 05:19:31 2021 +0000

    fixed make and something else probably

commit 90104035963fc6bd759e35cc2c5925fa54b992c5
Merge: bc4f1e9 343e262
Author: Ben <bs3148@columbia.edu>
Date:  Wed Mar 24 05:05:51 2021 +0000

    Merge remote-tracking branch 'origin/additional_tests'

commit bc4f1e93a7f8bbd12403ec1cc2d40056c48a6001
Author: Jason <jasoncardinale19@gmail.com>
Date:  Wed Mar 24 00:55:53 2021 -0400

    demo working on master

commit 5dc7c28a9cca36a319258737222519d33ee4b307
Author: evantilley <elt2141@columbia.edu>
Date:  Tue Mar 23 05:29:10 2021 +0000

    changed file endings to .bug and merged

commit 5577959d2007af8de8eb28335202f40d49ecaa34
Merge: e9b837f 4afb3be
Author: Ben <bs3148@columbia.edu>
Date:  Tue Mar 23 05:08:53 2021 +0000

    Merge branch 'string' of github.com:BenjaminSnyder/bugsy into string
```

```
commit e9b837fdd86e9601b31820b50f9163c5b76b7574
Author: Ben <bs3148@columbia.edu>
Date:   Tue Mar 23 05:08:49 2021 +0000

    updated to bugsy.native and makefile changes and working on improved
        compile script

commit 4afb3beacab4d3ce905b0c89902e6b17c7ab1409
Author: evantilley <elt2141@columbia.edu>
Date:   Tue Mar 23 04:56:48 2021 +0000

    added colors to test outputs + modified tests

commit e9c1a1cf1ed0449ced0d5befc44343e32ada64b7
Merge: 8575f7c 216dc75
Author: evantilley <elt2141@columbia.edu>
Date:   Tue Mar 23 04:54:21 2021 +0000

    Merge branch 'string' of https://github.com/BenjaminSnyder/bugsy
        into string

commit 216dc75d34640991889a7fff48987fe6073c50b7
Merge: 5d88fde 83027ce
Author: Ben <bs3148@columbia.edu>
Date:   Tue Mar 23 04:50:03 2021 +0000

    Merge branch 'string' of github.com:BenjaminSnyder/bugsy into string

commit 8575f7c79ed7ca15e45a548d04203e3972ced3d4
Merge: 83027ce dd935b7
Author: evantilley <elt2141@columbia.edu>
Date:   Tue Mar 23 04:42:28 2021 +0000

    Merge remote-tracking branch 'origin/string_test' into string

commit 83027ceae0a5cc5c927181331b0c14cb7e9521b5
Author: evantilley <elt2141@columbia.edu>
Date:   Tue Mar 23 04:41:28 2021 +0000

    removed quotes from strings when printed

commit dd935b7e555fdf4757138e66419f28ecff605c8a
Author: mwinitch <michaelwin26@gmail.com>
Date:   Tue Mar 23 04:36:34 2021 +0000

    Added testing and fixed parser error

commit 5d88fde807a983397f1e8431d70833a06dccf9ff
Author: Ben <bs3148@columbia.edu>
```

```
Date:  Tue Mar 23 03:56:25 2021 +0000

    renamed to bugsy

commit 38892bc8e718e5418c24df9578e768b4e1e7a38a
Merge: 99841d6 5fcb5d0
Author: mwinitch <michaelwin26@gmail.com>
Date:  Tue Mar 23 03:42:30 2021 +0000

    Merge branch 'string' into string_test

commit 99841d64c72148e9863fac9eee98e278b9c384d6
Author: mwinitch <michaelwin26@gmail.com>
Date:  Tue Mar 23 03:41:49 2021 +0000

    initial commit

commit 5fcb5d0a4c3cb984f1fbb49c29d07aadd3d31919
Author: evantilley <elt2141@columbia.edu>
Date:  Tue Mar 23 03:39:46 2021 +0000

    printing strings working!

commit 15f925367f8de4f0c8e2f141cbd1d96f48bece0c
Merge: 176a8c6 b5625ba
Author: Ben <bs3148@columbia.edu>
Date:  Tue Mar 23 03:31:10 2021 +0000

    Merge branch 'string' of github.com:BenjaminSnyder/bugsy into string

commit 176a8c63c7c2600008dc9d5f8638584037f3bc8b
Author: Ben <bs3148@columbia.edu>
Date:  Tue Mar 23 03:31:05 2021 +0000

    fixed strings scanner

commit b5625ba95790cf8bd245f3109aebe2e835c566fe
Author: Jason <jasoncardinale19@gmail.com>
Date:  Mon Mar 22 23:11:11 2021 -0400

    print for string

commit 277974e058273ea75f178715879607f7c72bcbc8
Author: evantilley <elt2141@columbia.edu>
Date:  Tue Mar 23 03:05:16 2021 +0000

    make works w/ strings

commit ce9a1712203d213ac8602d7846fd8597456ef90e
Merge: 0e58a47 06d37a9
```

```
Author: Ben <bs3148@columbia.edu>
Date:  Tue Mar 23 02:56:33 2021 +0000

    Merge remote-tracking branch 'origin/strings_codegen' into string

commit 06d37a978a6e1383341d0d96cb4612400617c97a
Author: evantilley <elt2141@columbia.edu>
Date:  Tue Mar 23 02:55:40 2021 +0000

    beginnings of strings for codegen

commit 0e58a475a585c50a965aa7e88b8e21153ad09d71
Author: Ben <bs3148@columbia.edu>
Date:  Tue Mar 23 02:53:54 2021 +0000

    fixed semant, bug with strlit vs string

commit 35d5f643d8b32e3e460e6c52535faf17961b9b3a
Author: Ben <bs3148@columbia.edu>
Date:  Tue Mar 23 02:48:24 2021 +0000

    fixed scanner, oops

commit 2fe601a1e280f68c2512b77158aa1ba3c455cf60
Merge: ff0c688 ed9b0e0
Author: Ben <bs3148@columbia.edu>
Date:  Tue Mar 23 02:42:20 2021 +0000

    Merge branch 'string' of github.com:BenjaminSnyder/bugsy into string

commit ff0c688e0523d8e67fab343337b45cd5d7288270
Author: Ben <bs3148@columbia.edu>
Date:  Tue Mar 23 02:41:35 2021 +0000

    added multiline comments and string support for scanner, parser, ast

commit ed9b0e0623a71ca7076545f8c73b62b4167339f3
Author: Jason <jasoncardinale19@gmail.com>
Date:  Mon Mar 22 22:41:51 2021 -0400

    Added string support for sast and semant

commit 564e9a714ef86829d71deffefe2df71b9bba2e37
Author: Ben <bs3148@columbia.edu>
Date:  Tue Mar 23 00:22:21 2021 +0000

    cleaning up dir and fix makefile

commit 5fa5ee4d37d9458383f61df6a8461cc7d8793940
Author: evantilley <elt2141@columbia.edu>
```

```
Date:  Sun Mar 21 22:32:58 2021 +0000

    partial hello world/nums mildly working now

commit d98229156e0dbdebf759d31fc302ac33cfab5891
Author: Jason <jasoncardinale19@gmail.com>
Date:  Sun Mar 21 17:47:29 2021 -0400

    merging semant changes from hello branch also with num and lit
        changes

commit a5b7ef87d23e50d998d1ab910df36684724c4071
Author: evantilley <elt2141@columbia.edu>
Date:  Sun Mar 21 20:36:15 2021 +0000

    working on fixing semant to get codegen working

commit 7d73fb21288b3c3974393c80f8c064c794a5d53d
Author: evantilley <elt2141@columbia.edu>
Date:  Sun Mar 21 06:11:25 2021 +0000

    big gains with codegen

commit 1654ee3f7ee6e4fb8542cdffbf9e59986a118d17
Merge: dfe3935 6efe6f1
Author: evantilley <elt2141@columbia.edu>
Date:  Sat Mar 20 18:05:55 2021 +0000

    Merge remote-tracking branch 'origin/flit' into real_codegen_evan

commit dfe3935fe4e75e8d729f171ae781ce348963752b
Author: evantilley <elt2141@columbia.edu>
Date:  Sat Mar 20 18:03:59 2021 +0000

    wip

commit 6efe6f1ef96ff9eed6a96b149bd2add75c612617
Merge: bb77c8e 685343a
Author: Ben <bs3148@columbia.edu>
Date:  Sat Mar 20 17:59:50 2021 +0000

    Merge branch 'flit' of github.com:BenjaminSnyder/bugsy into flit

commit bb77c8e53c50de706a644b7b63832fa6bc5614fa
Author: Ben <bs3148@columbia.edu>
Date:  Sat Mar 20 17:56:29 2021 +0000

    added original microc files and some fliteral and blit stuff

commit 685343ad5dd3c7a46e98fd241314d1c96fbf05af
```

```
Author: Ben <bs3148@columbia.edu>
Date:  Sat Mar 20 17:56:29 2021 +0000

    added original microc files and some fliteral and blit stuff

commit c02c506ab0594a5044b7e670f3d86644091648cb
Author: Ben <bs3148@columbia.edu>
Date:  Sat Mar 20 16:35:33 2021 +0000

    COMPILES

commit b25b99daf25a6cfd6020e79a97740fee584f4962
Author: mwinitch <michaelwin26@gmail.com>
Date:  Sat Mar 20 16:30:31 2021 +0000

    Removed array and tried to fix arithmetic operators

commit bd0927269342f0102a5f680079c9be9691d154b5
Author: mwinitch <michaelwin26@gmail.com>
Date:  Sat Mar 20 16:03:40 2021 +0000

    Added array functionality

commit d07e843252d95f9f8db020aab42727359fac45b1
Author: Ben <bs3148@columbia.edu>
Date:  Sat Mar 20 15:46:06 2021 +0000

    changed some array stuff

commit 95b14712d5050cb3485403f35c48545c6f19b549
Author: Ben <bs3148@columbia.edu>
Date:  Sat Mar 20 15:37:40 2021 +0000

    fixed codegen error, thanks hans

commit 82a6fc639ff0431a0b2458cf3e4774dda439a2a9
Author: Ben <bs3148@columbia.edu>
Date:  Thu Mar 18 22:07:30 2021 +0000

    removed endline whitespace

commit 7bee937eab3762bcddd33edb3369d49dca99bc01
Author: Ben <bs3148@columbia.edu>
Date:  Thu Mar 18 19:15:43 2021 +0000

    moved semant and codegen to a new folder orig_files and removed them
        from makefile

commit 4f688dfd0efe71ee8c0d13c8aee8df7ab64f8ecb
Merge: 5471626 f4146cb
```

```
Author: Ben <bs3148@columbia.edu>
Date:  Thu Mar 18 18:21:30 2021 +0000

    Merge branch 'master' of github.com:BenjaminSnyder/bugsy

commit 547162699f67ddfd471a46372bfd541345c9756e
Author: Ben <bs3148@columbia.edu>
Date:  Thu Mar 18 18:19:46 2021 +0000

    deleted unnecessary files

commit f4146cbabb6229fd679e7beb50f586bd143a6fd3
Merge: 732c75e 5472a14
Author: sofia <ss5664@columbia.edu>
Date:  Thu Mar 18 18:09:26 2021 +0000

    Merge branch 'master' into working

commit 5472a14028716ad69864dc93496a30f52c1cee9a
Merge: 59684f8 732c75e
Author: sofia <ss5664@columbia.edu>
Date:  Thu Mar 18 18:05:13 2021 +0000

    Merge remote-tracking branch 'origin/working'

commit 732c75e51c0cac45d6a5c048b1be5de7b459422f
Author: sofia <ss5664@columbia.edu>
Date:  Thu Mar 18 17:43:30 2021 +0000

    working

commit 343e262755b02714ecc0adbd039a12b963c4f529
Author: mwinitch <michaelwin26@gmail.com>
Date:  Wed Mar 17 04:10:06 2021 +0000

    Updated more tests

commit d913577831facc191cd78bfeb77106bc75379991
Author: Jason <jasoncardinale19@gmail.com>
Date:  Wed Mar 17 00:08:48 2021 -0400

    more fail test updates plus successes

commit 0037d2059325d9d8d6c8b113f0d312217e71b0b4
Author: Jason <jasoncardinale19@gmail.com>
Date:  Tue Mar 16 23:49:50 2021 -0400

    fail test mods

commit a8ea05498ab0a41662214260b69edb1abc93a8db
```

```
Author: Jason <jasoncardinale19@gmail.com>
Date:  Tue Mar 16 23:47:46 2021 -0400

    hi

commit 5ac7f36cb1165f41be801e7df2164d9a7d2ecbfa
Author: mwinitch <michaelwin26@gmail.com>
Date:  Wed Mar 17 03:39:12 2021 +0000

    Initial commit

commit 59684f81c4dfc7dfcc48e7df0d87d2fea3647835
Author: Ben <bs3148@columbia.edu>
Date:  Wed Mar 17 01:54:47 2021 +0000

    latest changes

commit 19f7a4c5608b138429cc50fcf7bc355862c61c7d
Merge: 690ef9e 35eef95
Author: Ben <bs3148@columbia.edu>
Date:  Sun Mar 14 20:35:23 2021 +0000

    Merge branch 'master' of https://github.com/BenjaminSnyder/bugsy

commit 690ef9ecb4237d012822abacc7795b8b3a5bf3be
Author: Ben <bs3148@columbia.edu>
Date:  Sun Mar 14 20:34:05 2021 +0000

    possibly fixed ast

commit 35eef95ec905fbbc5eb90d6ecba31b5cd866e4cd
Author: Benjamin Snyder <0701benjamin@gmail.com>
Date:  Thu Mar 11 14:33:24 2021 -0500

    Update README

commit 1267ed555789bed854ab8729f71b15e65a6262e7
Author: Ben <bs3148@columbia.edu>
Date:  Wed Mar 10 23:17:43 2021 +0000

    saving progress

commit 59a26252d8db59a5707644cf056774e19ef6949f
Author: Ben <bs3148@columbia.edu>
Date:  Wed Feb 24 00:32:49 2021 +0000

    added pattern matching probs broken to parser

commit 8e45479a4b037a85b765ccf18981551d0be15d4c
Author: Ben <bs3148@columbia.edu>
```

```
Date:  Tue Feb 23 02:23:40 2021 +0000

    added the rest of the tokens, need help on parser, asking hans
        tomorrow

commit 8817cb4888e54b21c22a4e0d64e2c2db8b883dca
Author: Ben <bs3148@columbia.edu>
Date:  Mon Feb 22 06:02:41 2021 +0000

    ast progress

commit 441b59f5e4df3b23a0a777b8378bae3cb2bbc793
Author: Ben <bs3148@columbia.edu>
Date:  Mon Feb 22 04:18:25 2021 +0000

    in theory added some support for classes and constructors and
        arrays???

commit d165f1b7515e7e3a41f9c6fb2be73db811e1d186
Author: Ben <bs3148@columbia.edu>
Date:  Mon Feb 22 03:30:50 2021 +0000

    initial types added to scanner .mll
```