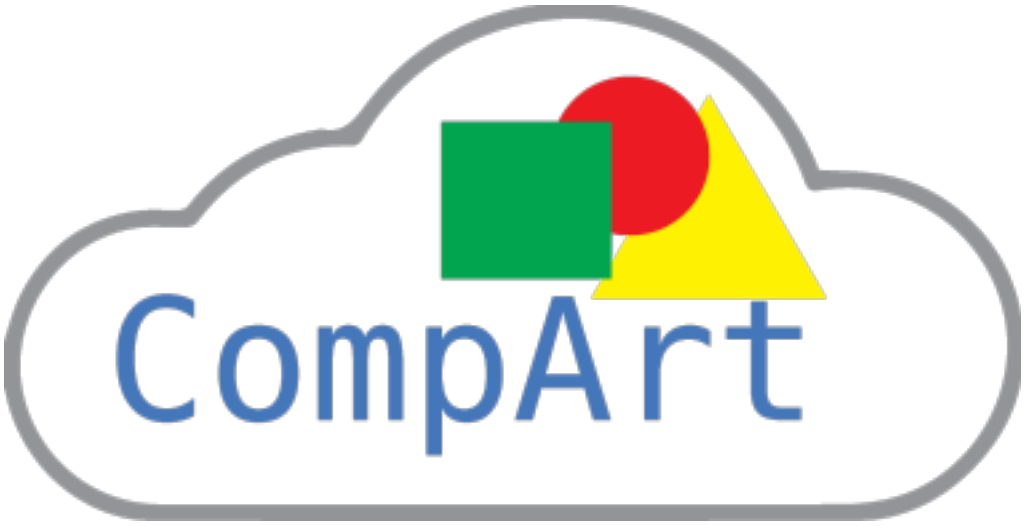
CompArt Final Report

Aaron Priven, Julia Reichel, Asher Willner, Evan Zauderer

Spring 2021

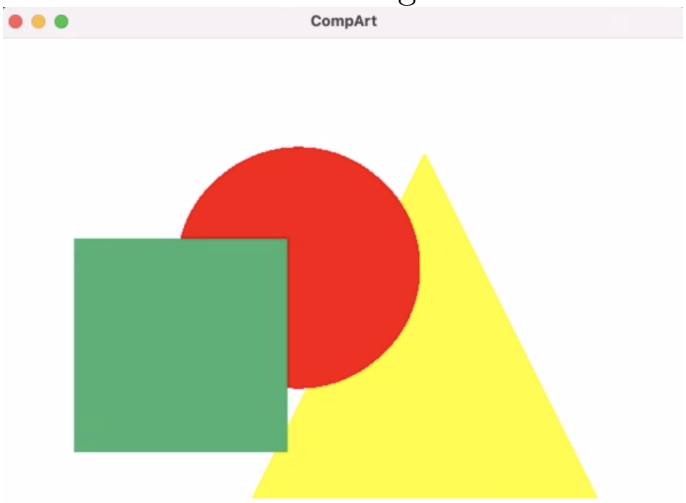


Sneak Peak:

From code...

```
1   int main(){
2      int i;
3      i = 0;
4      createWindow(600,600);
5      background(255,255,255);
6      for (;1<2;) {
7         color(255,255,0);
8         drawTriangle(220, 400, 370, 100, 520, 400);
9         color(255,0,0);
10        drawCircle(260,200,105);
11        color(60,179,113);
12        drawRect(65,175,250,360);
13        draw();
14        i = i + 1;
15     }
16     return 0;
17  }
```

...To logo



Contents

1 Introduction

1.1 Language White Paper

When we began this project, our group sought out a branch of computer science that we believed was needlessly overcomplicated. Our search led us to a very common, yet very difficult concept in computer science: drawing and animation. Libraries do exist for certain drawing capabilities, but the documentation can be extremely daunting for a task that one would likely claim that every learning programmer should be able to practice (since drawing allows the programmer to explore different implementations, to get the most beautiful outcome possible while building algorithms). Our project aimed to take on this task head-on, by simplifying the SDL family of libraries and incorporating it into our language. The syntax was meant to mimic some of the major programming languages available: notably C, and Java to some extent. However, the languages were stripped down to the parts that we believed would best allow for the user to implement these drawing functions. This is how CompArt was born.

Just a few months from start to finish, CompArt allows the user to use their imagination to dream up the best algorithms to draw and animate as they please. Additionally, the simplicity of the syntax allows a new learner to pick up the language quickly, and start showing just what CompArt can do. With its growing standard library, the power at the user's hands, alongside the quick learning curve the language provides, CompArt is a great way to learn to animate, as well as learn how to code, in a quick and enjoyable manner.

1.2 Goals

CompArt is meant to make computer artists' lives easier by providing them with an easy way to create beautiful digitale canvases. All an artist has to do is call upon functions that are built into our library (such as drawCircle(x,y,r)). The language is built on top of the Simple DirectMedia Layer (SDL2) Library and uses SDL's features, along with the SDL2 GFX extended library, in order to create a more friendly and comprehensible language.

2 Language Tutorial

2.1 Set up your Environment and Download Files

Download the zip file which contains all necessary files to use CompArt. Next install LLVM, OCaml, and OPAM

2.2 Compiling a Simple Program

Build compiler binary by running:
`$ make all`

Sometimes it is necessary to run this command beforehand:
`$ make clean`

To run a test, run:
`$ ./run.sh tests/test-mouse.mc`
(more generally: `tests/<filename>`)

2.3 Writing Simple Program

To create a file in CompArt, create a file with a .ca extension. Then inside that file, write source code following the details specified in our language reference manual. Save the file and run it by following the instructions here.

To compile and execute CompArt program into LLVM code:
`$ ./run.sh <filename>`

2.4 Debugging Program

To debug your program we have included print statements, such as:
`print()`

3 Language Reference Manual

3.1 Lexical Conventions

3.1.1 Tokens

There are five kinds of tokens described below: identifiers, keywords, constants, expression operators, and other separators. In general blanks, tabs, newlines, and comments are meant to separate tokens from one another.

3.1.2 Comments

Comments are introduced with the sequence of characters /* and terminated with the characters */. This implementation allows comments to span multiple lines.

3.1.3 Identifiers

An identifier is a sequence of letters, digits, and the underscore symbol. The first character of an identifier must be alphabetic and all letters and digits must be drawn from the Unicode character set. Upper and lower case letters are considered different and two identifiers are the same only if they have the same Unicode character for each letter or digit.

3.1.4 Keywords

The following identifiers are reserved for use as keywords, and may not be used otherwise:

1. int
2. bool
3. void
4. float
5. while
6. for
7. return
8. if
9. else
10. arr
11. main

3.2 Data Types

CompArt supports four fundamental types of objects: integers, floats, booleans, and arrays.

3.2.1 Integers

An integer is a signed sequence of digits. No decimal point is allowed in an integer. We incorporated int32.

3.2.2 Floats

A floating point number is a sequence of digits that includes a decimal point.

3.2.3 Booleans

A boolean is a single bit of data. Because of that, it can hold one of two values: True or False.

3.2.4 Arrays

An array is a list of elements. Arrays may only hold integers, the most practical use of arrays in CompArt.

3.3 Expressions

3.3.1 Precedence

The precedence of expression operators is that of highest precedence first. Within each subsection, the operators have the same precedence. Whether an operator has left or right associativity is specified in each subsection. If unspecified, the order of evaluation of expressions is undefined. This means that the compiler can freely choose to compute subexpressions in whatever order it believes is most efficient and fitting.

3.3.2 Primary Expressions

All primary expressions group left to right.

1. Identifier
   An identifier is a primary expression whose type is specified by its declaration.

2. (expression)
   A parenthesized expression is a primary expression whose type and value are identical to those of the unadorned expression. The presence of parentheses does not affect the primary expression.

3. primary-expression (expression-list)
   A function call is a primary expression followed by parentheses containing a possibly empty or comma-separated list of expressions that form the actual arguments to the function. The primary expression must be a type, and the result of the function call is that same type. Recursive calls to any function are permissible.

4. primary-value[expression]
   A subscripting expression contains a primary-value followed by a bracketed expression, which indicates the index of the desired element in the identified array. The index is required to be an integer between 0 and 1 less than the size of the array, or an error will be raised.

3.3.3 Unary operators

Note: All expressions with unary operators group right-to-left.

1. - expression
   The result is the negative of the expression, and has the same type. The type of the expression must be int or float.

2. ! expression
   The result of the logical negation operator ! is True if the value of the boolean expression is False, False if the value of the expression is True. The type of the result is a boolean. This operator is applicable to booleans and boolean expressions only.

3.3.4 Multiplicative operators

Note: The multiplicative operators * and / group left-to-right.

1. expression * expression
   The binary * operator indicates multiplication. If both operands are int, the result is int. If both operands are float, the result is float. No other combinations are allowed.

2. expression / expression
   The binary / operator indicates integer division. The same type considerations as for multiplication apply.

3.3.5 Additive operators

Note: The additive operators + and - group left-to-right.

1. expression + expression
   The result is the sum of the expressions. If both operands are int, the result is int. If both operands are float, the result is float. No other type combinations are allowed.

2. expression - expression
   The result is the difference of the operands. The same type considerations as for + apply.

3.3.6 Relational operators

Note: The relational operators group left-to-right, but this fact is not very useful; "a<b<c" does not mean what it seems to.

1. expression < expression
   The operator < (less than) yields True if the first expression yields a lesser value than the second expression. Else, it yields False. Operands may be ints or floats, as long as they match.

2. expression > expression
   The operator > (greater than) yields True if the first expression yields a greater value than the second expression. Else, it yields False. Operands may be ints or floats, as long as they match.

3. expression <= expression
   The operator <= (less than or equal to) yields True if the first expression yields a lesser value than the second expression OR they yield equal values. Else, it yields False. Operands may be ints or floats, as long as they match.

4. expression >= expression
   The operator >= (greater than or equal to) yields True if the first expression yields a greater value than the second expression OR they yield equal values. Else, it yields False. Operands may be ints or floats, as long as they match.

5. expression == expression
   The == (equal to) operator is exactly analogous to the relational operators except for its higher precedence here. (Thus `a<b == c<d` is 1 whenever a¡b and c¡d have the same truth-value). Any type is allowed for the operands, as long as both operands have the same type.

6. expression != expression
   The != (not equal to) operator is exactly analogous to the relational operators except for its higher precedence here. (Thus `a<b != c<d` is 1 whenever a<b and c<d have the opposite truth-value). Any type is allowed for the operands, as long as both operands have the same type.

7. expression && expression
   The && operator returns True if both its operands are True, False otherwise. && guarantees left-to-right evaluation; moreover the second operand is not evaluated if the first operand is False. The operands must be booleans.

8. expression || expression
   The || operator returns True if either of its operands are True, and False otherwise. || guarantees left-to-right evaluation; moreover, the second operand is not evaluated if the value of the first operand is True. The operands must be booleans.

3.3.7 Assignment operators

1. identifier = expression
   The assignment operator groups right-to-left. The value of the expression replaces that of the object referred to by the identifier. The types of the identifier and expression must be compatible.
   The value of the expression is simply stored into the object referred to by the identifier.

3.4 Statements

Except as indicated, statements are executed in sequence.

3.4.1 Expression statement

1. Most statements are expression statements, which have the form:
   `expression;`

2. Usually expression statements are assignments or function calls.

3.4.2 Block statement

1. Lists of statements can be defined in a specific block, delineated by braces {}.

3.4.3 Conditional statement

1. The two forms of the conditional statement are

   (a) `if ( expression ) statement;`

   (b) `if ( expression ) statement; else statement;`

2. In both cases the expression is evaluated and if it is True, the first sub-statement is executed. If the expression is False, the second sub-statement is executed. As usual, the "else" ambiguity is resolved by connecting an else with the last encountered elseless if. Note that the expression must be a boolean expression.

3.4.4 While statement

1. The while statement has the form:
   ```
   while ( expression ) { statement; }
   ```

2. The sub-statement is executed repeatedly so long as the value of the expression remains True. The test takes place before each execution of the statement. The expression must be a boolean expression

3.4.5 For statement

1. The for statement has the form:
   ```
   for ( expression-1opt ; expression-2 ; expression-3opt )
   {
        statement;
   }
   ```

2. This statement is equivalent to:
   ```
   expression-1opt;
   while ( expression-2) {
        statement;
        expression-3opt ;
   }
   ```

3. The first expression specifies initialization for the loop; the second specifies a test, made before each iteration, such that the loop is exited when the boolean expression becomes False; the third expression typically specifies an incrementation, or generally an update, which is performed after each iteration.

4. The 1st or 3rd expression may be dropped. They would be simply dropped from the expansion above.

3.5 Scope Rules

Any variable defined at the highest level of the program will be accessible throughout the program. If a variable is declared inside a function or a statement (i.e. a for-loop), it will only be accessible inside that function or statement.

3.6 Functions

3.6.1 Function declaration

A user can define a function with the following syntax:

```
return-type identifier ( { argument list} ) {
     { statement list }
     return expression; // unneeded if return-type is void
}
```

3.6.2 main() function

Whenever a CompArt program is ran, the computer will start by processing the main() function (after accessing all global variables). The main() function must return an int. Standard protocol is to return a 0, as a sign that the program ran successfully.

3.6.3 Using Functions

1. There is only one thing that can be done with a function after it is defined: call it. This is done as following:
   `identifier({argument list});`

2. If a function returns a value, it can be used locally or simply saved in a new local variable

3.7 Arrays and subscripting

3.7.1 Array Initialization

To initialize an array, one uses the notation:
`arr[size] identifier;`
to instantiate an array of the given size.
The length of the array is the maximum number of expressions in the list. The first element of the array is considered to be at position 0. An array must contain integers.

3.7.2 Editing Elements in Array

To edit elements of the array, one would index the array, as follows:
`identifier[index]=element;`

3.7.3 Accessing Elements in Array

To access elements of the array, one would index the array using subscripting, as follows:
`identifier[index];`

3.8 Standard Library

3.8.1 Description

The standard library will automatically be included for user convenience. This library includes functions that allows users to create different shapes, manipulate the colors and opacity of the shapes, specify the background color of the surface, and move their shapes with their computer mouse. The functions we defined call upon SDL2 and SDL_gfx functions.

3.8.2 Functions

These functions are all user defined functions that are defined in codegen.ml and can be found in compArtHelper.c

1. `int draw()`
    - displays the shapes that the user created on the SDL renderer
2. `int createWindow(int height, int width)`
    - sets up the SDL window surface and renderer
3. `int drawLine(int x1, int y1, int x2, int y2)`
    - creates a line from user specified coordinates
    - calls on the SDL function `lineRGBA(SDL surface, x1, y1, x2, y2, r, g, b, a)`
4. `int drawCircle(int x, int y, int r)`
    - creates a circle from user specified coordinates for a filled circle
    - calls on `filledCircleRGBA(SDL surface, x, y, rad, r, g, b, a)`
    - for an unfilled circle calls on `circleRGBA(SDL surface, x, y, rad, r, g, b, a)`
5. `int drawRect(int x, int y, int w, int h)`
    - creates a rectangle from user specified coordinates
    - for a filled rectangle calls on `boxRGBA(SDL surface, x1, y1, x2, y2, r, g, b, a)`
    - for an unfilled rectangle calls on `rectangleRGBA(SDL surface, x1, y1, x2, y2, r, g, b, a)`
6. `int drawTriangle(int x1, int y1, int x2, int y2, int x3, int y3)`
    - creates a triangle from user specified coordinates
    - for a filled triangle calls on `filledTrigonRGBA(x1, y1, x2, y2, x3, y3, r, g, b, a)`
    - for an unfilled triangle calls on `trigonRGBA(x1, y1, x2, y2, x3, y3, r, g, b, a)`
7. `color(int red, int green, int blue)`
    - changes color of the shape based on user specified RGB values
8. `void opacity(int x)`
    - sets the opacity value (a) equal to the user specified value

9. `void background(int red, int green, int blue)`
    - changes color of background based on user specified RGB values
10. `int getMouseX()`
    - gets the x coordinate position of the user's mouse
    - calls upon `SDL_GetMouseState(&mouseX, NULL)`
11. `int getMouseY()`
    - gets the y coordinate position of the user's mouse
    - calls upon `SDL_GetMouseState(NULL, &mouseY)`
12. `void fill()`
    - used to determine if object should be filled
13. `void noFill()`
    - used to determine if object should not be filled

4 Project Plan

4.1 Planning Process

From start to finish, the most important part of this project was the communication. Each week, our team met with our TA Wonhyuk (Harry) Choi to let us know how our progress was going, as well as provide feedback on our current product. In addition to these meetings, our group stayed in constant contact, often messaging each other in our text message group to make plans for additional weekly meetings. These meetings were often split into pairs (commonly Asher-Evan and Aaron-Julia), as we believed that the complexity of this project would best be taken on by pairs of individuals, so each pair of individuals could help each other get through the intricacies that OCaml and LLVM brought along. Therefore, at every one time there were two major tasks being taken on by two groups in parallel, on different Git branches. Every week before our TA meeting we would reconvene to discuss our updates, and gameplan for how to make the most of our valuable time with Harry. This weekly schedule was maintained throughout the project, and when paired with the checkpoints provided by the assignments, CompArt was being created on a clean and well-planned schedule.

As implementation went, we tried to stick to our pre-planned schedule as best as possible (seen below). This allowed us to make progress on our project, while making sure to keep the entire project in perspective, so we were able to modify and optimize our plan constantly. It led to our current implementation, where we focus largely on scalability. The current version of CompArt is amazing for anyone interested in getting started with drawing and animating through coding. However, we could add simple to use optimizations for this project in less than a day, due to the scalable setup of this project.

As functionality for the final product was added, so were test cases. We made sure to test every major addition, yielding a rich and diverse test suite that allowed us to find bugs in our code quickly so we could solve them right away. The simplicity of the syntax in our language allowed for these tests to be made quickly and often, which helped prove the great benefit of CompArt's syntax and entire setup from a developer's standpoint as well.

4.2 Programming Style Guide

We followed the following style guidelines throughout development:

- Consistent indentation to show dependencies
- CamelCase variables
- Keep lines short to ensure readability
- Use descriptive function names

- End each statement with a semicolon
- Remove all errors and warnings
- Commit frequently with meaningful messages
- Use of libraries when helpful
- CompArt files end in .ca

4.3 Project Timeline

1. Jan 20-25: Find a team and come up with a project idea
2. Jan 26-Feb 3: Work on the project proposal
3. Feb 4-24: Work on the LRM and Parser
4. Feb 25-March 24: Work on getting "Hello World" demo to run
5. March 25-April 14: Working on integrating SDL and gfx into our files, building out functions for our standard library
6. April 15-April 25: Work on implementing Arrays and creating test files
7. April 26: Final Presentation!

4.4 Roles and Responsibilities of Team Members

These were our initial roles (but they turned out to be very fluid):

Aaron Priven | Language Guru
Julia Reichel | Manager
Asher Willner | Tester
Evan Zauderer | System Architect

We found it more efficient to divide up and work in pairs.

- Pair #1: Asher Willner and Evan Zauderer worked mainly on the AST and implementing arrays
- Pair #2: Aaron Priven and Julia Reichel worked mainly on the integration of SDL2 and gfx into our language

4.5 Software Environment (Tools & Languages)

Libraries and Languages: C, Ocaml, Ocamlyacc, Ocammllex, LLVM, SDL2, and SDL2_gfx
Software: Visual Studio Code
OS: Mac OSX 11.2.3
Version Control: Github

4.6 Project Git Log

Note: the commits were mainly made by 2 individuals, as a result of our working in pairs dynamic mentioned above.

```
commit 33ac0aba64ad6feb82c7607f5e88fd8d4669ac00
Merge: b1ed52f 3e86a8b
Author: Aaron Priven <aaronhpriven@gmail.com>
Date:   Wed Apr 21 20:37:47 2021 -0400

    Merge branch 'main' of https://github.com/aaronhpriven/CompArt
    into main

commit b1ed52fc7a538406833bfe5351f10e261cdee23e
Author: Aaron Priven <aaronhpriven@gmail.com>
Date:   Wed Apr 21 20:37:40 2021 -0400

    updated files as of 4/21

commit 3e86a8bfd66dbdfc016792cf3100170f600744da
Author: asherwillner <asherjwillner@gmail.com>
Date:   Sat Apr 10 23:12:22 2021 -0700

    updated some tests

commit 404a0e17b0903ed9385f6f5d700b13ac9def5129
Merge: 6779ba5 fd87e3c
Author: asherwillner <asherjwillner@gmail.com>
Date:   Fri Apr 9 09:56:25 2021 -0700

    Merge branch 'main' of https://github.com/aaronhpriven/CompArt
    into main
    this one is necessary

commit 6779ba5dd32d2684c6f7586eb9c6cf035a2e1e03
Merge: 0a15db2 dc98fad
Author: asherwillner <asherjwillner@gmail.com>
Date:   Fri Apr 9 09:54:39 2021 -0700

    Merge branch 'main' of https://github.com/aaronhpriven/CompArt
    into main

    bec it's necessary

commit 0a15db20fb75aae4223de925c7246a37fe99097a
Author: asherwillner <asherjwillner@gmail.com>
Date:   Fri Apr 9 09:54:27 2021 -0700

    del

commit fd87e3c5f079eb265e261042eb63adc8a8dec437
Author: Aaron Priven <aaronhpriven@gmail.com>
Date:   Fri Apr 9 12:54:18 2021 -0400

    April 9th Commit

commit dc98fad3f60d0f203562b7ea1c9df4eb97cb2480
Author: Aaron Priven <aaronhpriven@gmail.com>
```

```
Date:   Wed Mar 24 12:44:11 2021 -0400

    progress 3/24 12:45

commit 8288c908ffd19ac0cf57d73bd39ac2e98cc01bdb
Author: Aaron Priven <aaronhpriven@gmail.com>
Date:   Tue Mar 23 21:01:49 2021 -0400

    Added docker set-up with edits from 3/23

commit 9f1d37a96a2c790ab1f41552fde153799c7ea1c6
Author: asherwillner <asherjwillner@gmail.com>
Date:   Tue Mar 23 17:15:54 2021 -0400

    our stuff

commit a5ba44ac3d54519f04b517264c6f7be77bd36ebc
Author: asherwillner <asherjwillner@gmail.com>
Date:   Sun Mar 21 11:25:13 2021 -0400

    update

commit 6520c126b6f5a88cac0e87e0dc1f3229bbcef785
Author: asherwillner <asherjwillner@gmail.com>
Date:   Sun Mar 21 11:06:46 2021 -0400

    project code folder

commit 4fa32ae1b7b220c91273e072b865f47ad1d49290
Merge: d8556c3 1ef0dfc
Author: asherwillner <asherjwillner@gmail.com>
Date:   Wed Feb 24 14:02:46 2021 -0500

    updated new struct

commit d8556c3cb19cb6ff0967ae860c2c61d7e00ebf7c
Author: asherwillner <asherjwillner@gmail.com>
Date:   Wed Feb 24 14:00:03 2021 -0500

    updated new struct

commit 1ef0dfc59ddad3353919612d243b56f8892f12ba
Author: aaronhpriven <69937659+aaronhpriven@users.noreply.github.
    com>
Date:   Tue Feb 23 19:52:55 2021 -0500

    Made minor adjustments to the comment lines

commit 4e4dc8ad9300e7ddb2897722d066a891919f5c9f
Author: asherwillner <asherjwillner@gmail.com>
Date:   Tue Feb 23 19:44:10 2021 -0500

    fixed reduce

commit d6a31f719f7ac0ec9149d1751cc3d46f1f18cef6
Author: asherwillner <asherjwillner@gmail.com>
Date:   Tue Feb 23 19:20:34 2021 -0500
```

```
107
108     updated comments
109
110 commit 3d93ca9511f18568fa65b8376326a3f3ef235c8b
111 Author: Aaron Priven <aaronhpriven@gmail.com>
112 Date:   Tue Feb 23 18:49:09 2021 -0500
113
114     Initial Commit
```

```
1  commit 180388730f9bf69254700162411f519a78f73f45
2  Merge: bc2a38b 250c025
3  Author: Aaron Priven <aaronhpriven@gmail.com>
4  Date:   Mon Apr 26 14:33:30 2021 -0400
5
6      Merge branch 'master' of https://github.com/aaronhpriven/
       CompArt2.0 into main
7
8  commit 250c025eb81ba441a1f5c1253bb4a7089a54167f
9  Author: asherwillner <asherjwillner@gmail.com>
10 Date:   Mon Apr 26 14:33:03 2021 -0400
11
12     delete extra code
13
14 commit bc2a38bbdbd2f688db31c30adcf2555e2e7b06b4
15 Author: Aaron Priven <aaronhpriven@gmail.com>
16 Date:   Mon Apr 26 14:32:47 2021 -0400
17
18     multiple balls using array test
19
20 commit 167cbd6625cba8d2ed1d9714f3d9138402501176
21 Merge: bac91fd ab63bb8
22 Author: Aaron Priven <aaronhpriven@gmail.com>
23 Date:   Mon Apr 26 14:08:02 2021 -0400
24
25     Merge branch 'master' of https://github.com/aaronhpriven/
       CompArt2.0 into main
26
27 commit ab63bb8c12c76830623272737b254ad7275828a9
28 Author: asherwillner <asherjwillner@gmail.com>
29 Date:   Mon Apr 26 14:06:48 2021 -0400
30
31     allowing symbols for arrays
32
33 commit bac91fd3b0c73d2447b97d15c2b680342ed307fc
34 Author: Aaron Priven <aaronhpriven@gmail.com>
35 Date:   Mon Apr 26 13:47:44 2021 -0400
36
37     comments deleted
38
39 commit 52b8396f8aa5dd80f8864075ec71d0dd336cbdf4
40 Author: Aaron Priven <aaronhpriven@gmail.com>
41 Date:   Mon Apr 26 13:30:50 2021 -0400
42
43     mouse game test added
44
45 commit f809db897f7097537d07003319e0289f214e5ee9
46 Merge: 1b74afe 6b013be
47 Author: Aaron Priven <aaronhpriven@gmail.com>
```

```
Date:   Mon Apr 26 12:42:40 2021 -0400

    Merge branch 'main' of https://github.com/aaronhpriven/CompArt2
    .0 into main

commit 1b74afe91f43a16ff34246fa4ee39e4e8e095e33
Author: Aaron Priven <aaronhpriven@gmail.com>
Date:   Mon Apr 26 12:13:39 2021 -0400

    renamed file exts. and added run.sh script

commit 6b013be029f7f2f1c636fb3f76b413a2b4a14ba1
Merge: ef97b75 2ae37ba
Author: Julia Reichel <jrr2189@barnard.edu>
Date:   Mon Apr 26 00:54:59 2021 -0400

    Deleted commented code.

commit 2ae37ba0bfe5e5deccc103c81338549a5ce310f7
Author: Aaron Priven <aaronhpriven@gmail.com>
Date:   Mon Apr 26 00:06:40 2021 -0400

    mouse functionality working

commit 50550b13bc0e7a9050b01bdd3fd8fb41ae4ad515
Author: Aaron Priven <aaronhpriven@gmail.com>
Date:   Sun Apr 25 23:41:35 2021 -0400

    files renamed

commit 94ac9595ca3f523472593eb591cf313cdf7a8205
Author: Aaron Priven <aaronhpriven@gmail.com>
Date:   Sun Apr 25 23:23:07 2021 -0400

    pre-renaming

commit 8b86fa9ec680d56aed0f10d5a41aeeba3263a163
Merge: 57624c1 e739c33
Author: Aaron Priven <aaronhpriven@gmail.com>
Date:   Sun Apr 25 22:13:14 2021 -0400

    Merge branch 'master' of https://github.com/aaronhpriven/
    CompArt2.0 into main

commit ef97b75564b24760b8a7a64e3b9a0db4d37ec4aa
Merge: 3d265f1 57624c1
Author: Julia Reichel <jrr2189@barnard.edu>
Date:   Sun Apr 25 22:12:13 2021 -0400

    Merge branch 'main' of https://github.com/aaronhpriven/CompArt2
    .0 into main

commit 57624c158eeb56a03982b88169bab0dd91979372
Merge: 52283a1 4a3162b
Author: aaronhpriven <69937659+aaronhpriven@users.noreply.github.
    com>
Date:   Sun Apr 25 22:12:00 2021 -0400
```

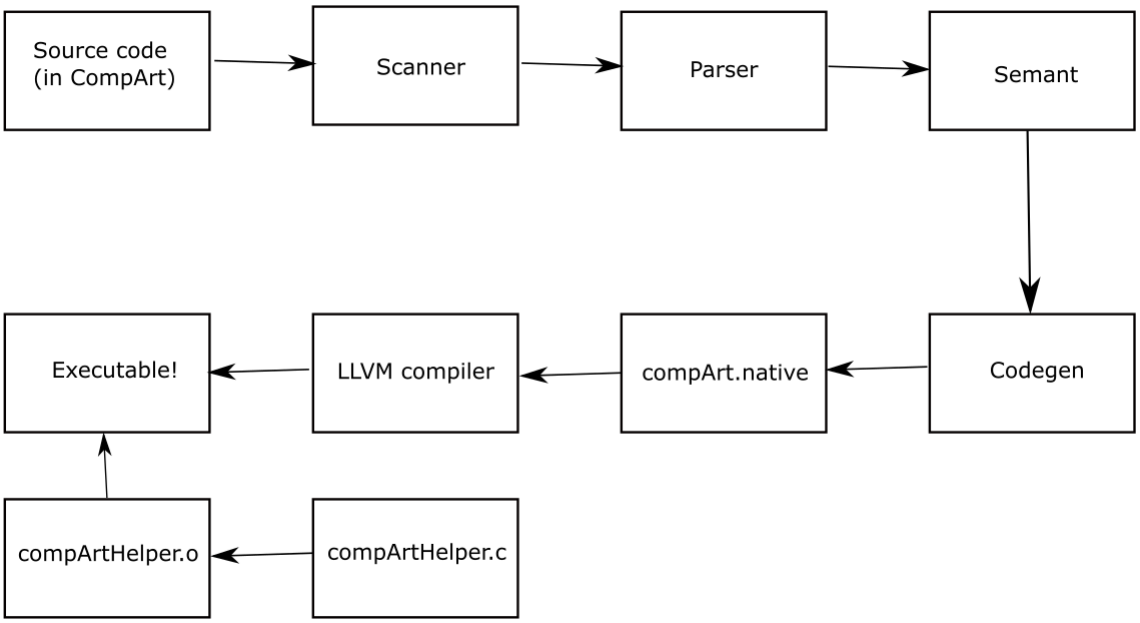
```
101
102     Merge pull request #2 from aaronhpriven/aaron
103
104     Aaron
105
106 commit 4a3162bd98ae90be2213e89516cd914e8b71dff6
107 Author: Aaron Priven <aaronhpriven@gmail.com>
108 Date:   Sun Apr 25 22:10:47 2021 -0400
109
110     ready for pull
111
112 commit 3d265f13a23d04608ac73b47bc5c1d7d46e4821b
113 Merge: 52283a1 e739c33
114 Author: Julia Reichel <jrr2189@barnard.edu>
115 Date:   Sun Apr 25 21:53:26 2021 -0400
116
117     Merge branch 'master' of https://github.com/aaronhpriven/
    CompArt2.0 into main
118
119 commit e739c330a1b2a1d0e2a962dda72cbafea2e517aa
120 Author: asherwillner <asherjwillner@gmail.com>
121 Date:   Sun Apr 25 21:52:23 2021 -0400
122
123     cleaned up code
124
125 commit f918757eef690257c350d6ac0feb6944bd379d8b
126 Author: asherwillner <asherjwillner@gmail.com>
127 Date:   Sun Apr 25 21:50:33 2021 -0400
128
129     with arrays and with commented out stuff to make arrays into
    pointers
130
131 commit ca5c2dcce4064aa5c9b09109e590891bb0e8f0b8
132 Author: Aaron Priven <aaronhpriven@gmail.com>
133 Date:   Sun Apr 25 17:06:51 2021 -0400
134
135     renderer clears after each cycle of draw()
136
137 commit 86133a47e601daabb0d6ca8b57acdcab66b06a28
138 Author: Aaron Priven <aaronhpriven@gmail.com>
139 Date:   Sun Apr 25 13:18:48 2021 -0400
140
141     bunch of tests working
142
143 commit ab072e906b3463cab529d3e9ca9d3f6885958978
144 Author: Aaron Priven <aaronhpriven@gmail.com>
145 Date:   Sun Apr 25 12:38:30 2021 -0400
146
147     .
148
149 commit 7ff581b286e084819e5e2eff6c64f32edd79d069
150 Author: Aaron Priven <aaronhpriven@gmail.com>
151 Date:   Sun Apr 25 12:37:45 2021 -0400
152
153     createWindow(), fill, nofill...
154
155 commit fb3e2e153bfbaa0821f54a649667b6f80b6c4d9c
```

```
Author: Aaron Priven <aaronhpriven@gmail.com>
Date:   Sat Apr 24 22:10:19 2021 -0400

    solved issue of gfx error about surface/renderer

commit 52283a1dbb9fd27334837e46c42096d25cdcecf3
Merge: 0092c8e cc17704
Author: aaronhpriven <69937659+aaronhpriven@users.noreply.github.
    com>
Date:   Sat Apr 24 21:31:31 2021 -0400

    Merge pull request #1 from aaronhpriven/aaron

    Aaron

commit cc17704e865c7cc448ecf6b726c83636cca791d1
Author: Aaron Priven <aaronhpriven@gmail.com>
Date:   Fri Apr 23 17:31:27 2021 -0400

    multiple params working. run 'make movingball'

commit e5adb73ab03fdb2346f7e35f9ca2d1aa0154c8d7
Author: Aaron Priven <aaronhpriven@gmail.com>
Date:   Fri Apr 23 16:38:10 2021 -0400

    Line with 4 params working. Draw works with no param

commit 813a257424dc1c9970d0c90a190afef92c727571
Author: Aaron Priven <aaronhpriven@gmail.com>
Date:   Fri Apr 23 15:17:57 2021 -0400

    color, circle (gfx) working

commit 55ad6830dba24f1edf3794a08477442502c9e014
Author: Aaron Priven <aaronhpriven@gmail.com>
Date:   Fri Apr 23 10:06:19 2021 -0400

    added color function. run 'make color'

commit 5c8453eaa2b284938aaec244873eac3ed9580842
Author: Aaron Priven <aaronhpriven@gmail.com>
Date:   Fri Apr 23 09:36:47 2021 -0400

    created test-setupdraw which draws many rectangles to the
    screen. This commit adds a setup and draw function. Run 'make
    setupdraw' to run it

commit 0092c8eef9c17481daa8b075eb18ab8093a09a89
Author: Aaron Priven <aaronhpriven@gmail.com>
Date:   Thu Apr 22 21:38:23 2021 -0400

    draw function from microc succesfully calls draw in
    testdrawchessboard.c. Run 'make draw' to see some magic! (I
    added a draw script that builds all the necessary files and
    executes draw, a chessboard should appear. After closing it,
    all files will be removed

```

```
commit 21dbbf31100ba2cfbe1b76b535b56fc7d90026e0
Author: Aaron Priven <aaronhpriven@gmail.com>
Date:   Thu Apr 22 10:17:23 2021 -0400

    added chess board

commit 9ee74f5121e9e9fc31c2c2fe0e996c0bab57b88b
Author: Aaron Priven <aaronhpriven@gmail.com>
Date:   Thu Apr 22 10:04:39 2021 -0400

    initial commit
```

5 Architectural Design

5.1 Block Diagrams



5.2 Description of Interfaces between Components

The scanner and parser were tackled as a team, as well as Semant and Codegen (since both groups' work was involved in both Semant and Codegen, especially Asher and Evan's). Outside of this, Evan and Asher put more effort into the AST and SAST, while Julia and Aaron took on compArtHelper.c.

5.2.1 Source Code

This is the program that the user will be writing in CompArt. In practice, all CompArt files should in .ca

5.2.2 Scanner (aka: lexer)

The scanner (scanner.mll) takes in a CompArt source program of ASCII characters and translates them into tokens. If any illegal characters are found, lexing errors will be thrown.

5.2.3 Parser

The parser (parser.mly) converts the tokens produced by the scanner into an abstract syntax tree (AST) based on the CompArt syntax rules described in the Language Reference Manual (LRM). If syntax errors are found, parser errors will be thrown.

5.2.4 Semantic Checker

The semantic checker (semant.ml) converts the AST into a semantically-checked abstract syntax tree (SAST).

5.2.5 Code Generation

The code generation (codegen.ml) traverses through the SAST and produces LLVM code.

5.2.6 compArt.native

This is the interpreter that runs the .ca files and generates the *.ll files.

5.2.7 LLVM compiler

The LLVM compiler converts the input LLVM into assembly language, ready to be passed to the executable

5.2.8 compArtHelper.c

This C file is the way that CompArt integrates with SDL. In this file we incorporate and intertwine many SDL functions and types in order to create simple function calls for the user in CompArt.

5.2.9 compArtHelper.o

This .o file is the binary code created by the compilation process.

5.2.10 Executable

Through the process of linking the .o file and supporting code, this executable is created.

6 Test Plan

6.1 Testing Suite and Automation

All of our tests are stored in the /test/ folder. Success tests are of the form `test-*.ca` and fail tests are of the form `fail-*.ca`
Testing automation is completed by simply running:
`$./run.sh tests/{test-name}.ca`

6.2 Unit and Integration Testing

We decided to create tests often, with the creation of each new feature. These tests can be seen in the Appendix (in the Tests section. The ones written above are omitted.)

6.3 Examples of Source Language Programs

Mouse tracking example:

```
int main() {
   int i;
   int x;
   int y;
   int r;
   int xspeed;
   int yspeed;
   int window_w;
   int window_h;
   int mouseX;
   int mouseY;
   int block_w;
   int block_h;
   int timer;
   int red;

   x = 100;
   y = 300;
   xspeed = 1;
   yspeed = 1;
   window_w = 600;
   window_h = 600;
   block_w = 80;
   block_h = 10;
   timer = 0;
   red = 0;
   r = 20;

   createWindow(window_w, window_h);

   for (;1<2;i = i + 1) {
      mouseX = getMouseX();
      mouseY = getMouseY();
      timer = timer - 1;
      if(timer > 0) red = 1; else red = 0;
```

```
      color(x, y, y-x);
      if (red == 1) background(255,0,0); else background
    (100,100,100);


      drawRect(mouseX-(block_w/2), window_h-block_h, mouseX+(
    block_w/2), window_h);

      color(0, 0, 0);
      drawCircle(x, y, r);
      if (x > window_w-r) xspeed = -xspeed;
      if (x < r) xspeed = -xspeed;


      if (y > (window_h-(r+block_h)))
         if((mouseX-x) < block_w/2)
            if((mouseX-x) > -(block_w/2))
               yspeed = -yspeed;

      if (y > (window_h+r))
         {y = 100;
         timer = 50;
         }

      if (y < r) yspeed = -yspeed;
      x = x + xspeed;
      y = y + yspeed;
      draw();
   }
   return 0;
}
```

```
; ModuleID = 'compArt'
source_filename = "compArt"

@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

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

declare i32 @printbig(i32)

declare i32 @draw()

declare i32 @createWindow(i32, i32)

declare i32 @background(i32, i32, i32)

declare i32 @color(i32, i32, i32)

declare i32 @opacity(i32)

declare i32 @fill()

declare i32 @noFill()

declare i32 @drawRect(i32, i32, i32, i32)
```

```
26
27 declare i32 @drawLine(i32, i32, i32, i32)
28
29 declare i32 @drawCircle(i32, i32, i32)
30
31 declare i32 @drawTriangle(i32, i32, i32, i32, i32, i32)
32
33 declare i32 @getMouseX()
34
35 declare i32 @getMouseY()
36
37 define i32 @main() {
38 entry:
39   %i = alloca i32, align 4
40   %x = alloca i32, align 4
41   %y = alloca i32, align 4
42   %r = alloca i32, align 4
43   %xspeed = alloca i32, align 4
44   %yspeed = alloca i32, align 4
45   %window_w = alloca i32, align 4
46   %window_h = alloca i32, align 4
47   %mouseX = alloca i32, align 4
48   %mouseY = alloca i32, align 4
49   %block_w = alloca i32, align 4
50   %block_h = alloca i32, align 4
51   %timer = alloca i32, align 4
52   %red = alloca i32, align 4
53   store i32 100, i32* %x, align 4
54   store i32 300, i32* %y, align 4
55   store i32 3, i32* %xspeed, align 4
56   store i32 3, i32* %yspeed, align 4
57   store i32 600, i32* %window_w, align 4
58   store i32 600, i32* %window_h, align 4
59   store i32 80, i32* %block_w, align 4
60   store i32 10, i32* %block_h, align 4
61   store i32 0, i32* %timer, align 4
62   store i32 0, i32* %red, align 4
63   store i32 20, i32* %r, align 4
64   %window_h1 = load i32, i32* %window_h, align 4
65   %window_w2 = load i32, i32* %window_w, align 4
66   %createWindow = call i32 @createWindow(i32 %window_w2, i32 %
       window_h1)
67   br label %while
68
69 while:                                           ; preds = %
       merge93, %entry
70   br i1 true, label %while_body, label %merge106
71
72 while_body:                                      ; preds = %while
73   %getMouseX = call i32 @getMouseX()
74   store i32 %getMouseX, i32* %mouseX, align 4
75   %timer3 = load i32, i32* %timer, align 4
76   %tmp = sub i32 %timer3, 1
77   store i32 %tmp, i32* %timer, align 4
78   %timer4 = load i32, i32* %timer, align 4
79   %tmp5 = icmp sgt i32 %timer4, 0
80   br i1 %tmp5, label %then, label %else
```

```
81
82  merge:                                            ; preds = %else,
        %then
83    %y6 = load i32, i32* %y, align 4
84    %x7 = load i32, i32* %x, align 4
85    %tmp8 = sub i32 %y6, %x7
86    %y9 = load i32, i32* %y, align 4
87    %x10 = load i32, i32* %x, align 4
88    %color = call i32 @color(i32 %x10, i32 %y9, i32 %tmp8)
89    %red11 = load i32, i32* %red, align 4
90    %tmp12 = icmp eq i32 %red11, 1
91    br i1 %tmp12, label %then14, label %else15
92
93  then:                                             ; preds = %
        while_body
94    store i32 1, i32* %red, align 4
95    br label %merge
96
97  else:                                             ; preds = %
        while_body
98    store i32 0, i32* %red, align 4
99    br label %merge
100
101 merge13:                                          ; preds = %else15
        , %then14
102   %window_h17 = load i32, i32* %window_h, align 4
103   %mouseX18 = load i32, i32* %mouseX, align 4
104   %block_w19 = load i32, i32* %block_w, align 4
105   %tmp20 = sdiv i32 %block_w19, 2
106   %tmp21 = add i32 %mouseX18, %tmp20
107   %window_h22 = load i32, i32* %window_h, align 4
108   %block_h23 = load i32, i32* %block_h, align 4
109   %tmp24 = sub i32 %window_h22, %block_h23
110   %mouseX25 = load i32, i32* %mouseX, align 4
111   %block_w26 = load i32, i32* %block_w, align 4
112   %tmp27 = sdiv i32 %block_w26, 2
113   %tmp28 = sub i32 %mouseX25, %tmp27
114   %drawRect = call i32 @drawRect(i32 %tmp28, i32 %tmp24, i32 %tmp21
        , i32 %window_h17)
115   %color29 = call i32 @color(i32 0, i32 0, i32 0)
116   %r30 = load i32, i32* %r, align 4
117   %y31 = load i32, i32* %y, align 4
118   %x32 = load i32, i32* %x, align 4
119   %drawCircle = call i32 @drawCircle(i32 %x32, i32 %y31, i32 %r30)
120   %x33 = load i32, i32* %x, align 4
121   %window_w34 = load i32, i32* %window_w, align 4
122   %r35 = load i32, i32* %r, align 4
123   %tmp36 = sub i32 %window_w34, %r35
124   %tmp37 = icmp sgt i32 %x33, %tmp36
125   br i1 %tmp37, label %then39, label %else42
126
127 then14:                                           ; preds = %merge
128   %background = call i32 @background(i32 255, i32 0, i32 0)
129   br label %merge13
130
131 else15:                                           ; preds = %merge
132   %background16 = call i32 @background(i32 100, i32 100, i32 100)
```

```
133   br label %merge13
134
135 merge38:                                          ; preds = %else42
      , %then39
136   %x43 = load i32, i32* %x, align 4
137   %r44 = load i32, i32* %r, align 4
138   %tmp45 = icmp slt i32 %x43, %r44
139   br i1 %tmp45, label %then47, label %else50
140
141 then39:                                           ; preds = %
      merge13
142   %xspeed40 = load i32, i32* %xspeed, align 4
143   %tmp41 = sub i32 0, %xspeed40
144   store i32 %tmp41, i32* %xspeed, align 4
145   br label %merge38
146
147 else42:                                           ; preds = %
      merge13
148   br label %merge38
149
150 merge46:                                          ; preds = %else50
      , %then47
151   %y51 = load i32, i32* %y, align 4
152   %window_h52 = load i32, i32* %window_h, align 4
153   %r53 = load i32, i32* %r, align 4
154   %block_h54 = load i32, i32* %block_h, align 4
155   %tmp55 = add i32 %r53, %block_h54
156   %tmp56 = sub i32 %window_h52, %tmp55
157   %tmp57 = icmp sgt i32 %y51, %tmp56
158   br i1 %tmp57, label %then59, label %else81
159
160 then47:                                           ; preds = %
      merge38
161   %xspeed48 = load i32, i32* %xspeed, align 4
162   %tmp49 = sub i32 0, %xspeed48
163   store i32 %tmp49, i32* %xspeed, align 4
164   br label %merge46
165
166 else50:                                           ; preds = %
      merge38
167   br label %merge46
168
169 merge58:                                          ; preds = %else81
      , %merge66
170   %y82 = load i32, i32* %y, align 4
171   %window_h83 = load i32, i32* %window_h, align 4
172   %r84 = load i32, i32* %r, align 4
173   %tmp85 = add i32 %window_h83, %r84
174   %tmp86 = icmp sgt i32 %y82, %tmp85
175   br i1 %tmp86, label %then88, label %else89
176
177 then59:                                           ; preds = %
      merge46
178   %mouseX60 = load i32, i32* %mouseX, align 4
179   %x61 = load i32, i32* %x, align 4
180   %tmp62 = sub i32 %mouseX60, %x61
181   %block_w63 = load i32, i32* %block_w, align 4
```

```
  %tmp64 = sdiv i32 %block_w63, 2
  %tmp65 = icmp slt i32 %tmp62, %tmp64
  br i1 %tmp65, label %then67, label %else80

merge66:                                          ; preds = %else80
    , %merge75
  br label %merge58

then67:                                           ; preds = %then59
  %mouseX68 = load i32, i32* %mouseX, align 4
  %x69 = load i32, i32* %x, align 4
  %tmp70 = sub i32 %mouseX68, %x69
  %block_w71 = load i32, i32* %block_w, align 4
  %tmp72 = sdiv i32 %block_w71, 2
  %tmp73 = sub i32 0, %tmp72
  %tmp74 = icmp sgt i32 %tmp70, %tmp73
  br i1 %tmp74, label %then76, label %else79

merge75:                                          ; preds = %else79
    , %then76
  br label %merge66

then76:                                           ; preds = %then67
  %yspeed77 = load i32, i32* %yspeed, align 4
  %tmp78 = sub i32 0, %yspeed77
  store i32 %tmp78, i32* %yspeed, align 4
  br label %merge75

else79:                                           ; preds = %then67
  br label %merge75

else80:                                           ; preds = %then59
  br label %merge66

else81:                                           ; preds = %
    merge46
  br label %merge58

merge87:                                          ; preds = %else89
    , %then88
  %y90 = load i32, i32* %y, align 4
  %r91 = load i32, i32* %r, align 4
  %tmp92 = icmp slt i32 %y90, %r91
  br i1 %tmp92, label %then94, label %else97

then88:                                           ; preds = %
    merge58
  store i32 100, i32* %y, align 4
  store i32 50, i32* %timer, align 4
  br label %merge87

else89:                                           ; preds = %
    merge58
  br label %merge87

merge93:                                          ; preds = %else97
    , %then94
```

```
  %x98 = load i32, i32* %x, align 4
  %xspeed99 = load i32, i32* %xspeed, align 4
  %tmp100 = add i32 %x98, %xspeed99
  store i32 %tmp100, i32* %x, align 4
  %y101 = load i32, i32* %y, align 4
  %yspeed102 = load i32, i32* %yspeed, align 4
  %tmp103 = add i32 %y101, %yspeed102
  store i32 %tmp103, i32* %y, align 4
  %draw = call i32 @draw()
  %i104 = load i32, i32* %i, align 4
  %tmp105 = add i32 %i104, 1
  store i32 %tmp105, i32* %i, align 4
  br label %while

then94:                                           ; preds = %
    merge87
  %yspeed95 = load i32, i32* %yspeed, align 4
  %tmp96 = sub i32 0, %yspeed95
  store i32 %tmp96, i32* %yspeed, align 4
  br label %merge93

else97:                                           ; preds = %
    merge87
  br label %merge93

merge106:                                         ; preds = %while
  ret i32 0
}
```

Array example:

```
/*
./microc.native tests/fail-arrayglobal.mc > fail-arrayglobal.ll
Fatal error: exception Failure("illegal array in global context
    global myarr")
*/

/* arr[10] myarr; */

arr[10] passarr(arr[10] arr10) {
    return arr10;
}

arr[10] getarr() {
    arr[10] myarr;
    myarr[0] = 16;
    myarr[8] = 120;
    print(myarr[8]);
    return myarr;
}


int main() {
    arr[100] myarr;
    arr[10] newarr;
    myarr[5] = 10;
    myarr[2] = 12;
    newarr = getarr();
```

```
    newarr = passarr(newarr);
    print(newarr[0]);
    print(myarr[3]);
    print(myarr[2]=12);
    return 0;
}
```

```
; ModuleID = 'compArt'
source_filename = "compArt"

@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"%d\0A\00", align
    1
@fmt.3 = private unnamed_addr constant [4 x i8] c"%g\0A\00", align
    1
@fmt.4 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align
    1
@fmt.5 = private unnamed_addr constant [4 x i8] c"%g\0A\00", align
    1

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

declare i32 @printbig(i32)

declare i32 @draw()

declare i32 @createWindow(i32, i32)

declare i32 @background(i32, i32, i32)

declare i32 @color(i32, i32, i32)

declare i32 @opacity(i32)

declare i32 @fill()

declare i32 @noFill()

declare i32 @drawRect(i32, i32, i32, i32)

declare i32 @drawLine(i32, i32, i32, i32)

declare i32 @drawCircle(i32, i32, i32)

declare i32 @drawTriangle(i32, i32, i32, i32, i32, i32)

declare i32 @getMouseX()

declare i32 @getMouseY()

define i32 @main() {
entry:
  %myarr = alloca [100 x i32], align 4
  %newarr = alloca [10 x i32], align 4
  %_gep = getelementptr [100 x i32], [100 x i32]* %myarr, i32 0,
    i32 5
```

```
  store i32 10, i32* %_gep, align 4
  %_gep1 = getelementptr [100 x i32], [100 x i32]* %myarr, i32 0, i32 2
  store i32 12, i32* %_gep1, align 4
  %getarr_result = call [10 x i32] @getarr()
  store [10 x i32] %getarr_result, [10 x i32]* %newarr, align 4
  %newarr2 = load [10 x i32], [10 x i32]* %newarr, align 4
  %passarr_result = call [10 x i32] @passarr([10 x i32] %newarr2)
  store [10 x i32] %passarr_result, [10 x i32]* %newarr, align 4
  %_gep3 = getelementptr [10 x i32], [10 x i32]* %newarr, i32 0, i32 0
  %gep = load i32, i32* %_gep3, align 4
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32 %gep)
  %_gep4 = getelementptr [100 x i32], [100 x i32]* %myarr, i32 0, i32 5
  %gep5 = load i32, i32* %_gep4, align 4
  %printf6 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32 %gep5)
  %_gep7 = getelementptr [100 x i32], [100 x i32]* %myarr, i32 0, i32 2
  store i32 12, i32* %_gep7, align 4
  %printf8 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32 12)
  ret i32 0
}

define [10 x i32] @getarr() {
entry:
  %myarr = alloca [10 x i32], align 4
  %_gep = getelementptr [10 x i32], [10 x i32]* %myarr, i32 0, i32 0
  store i32 16, i32* %_gep, align 4
  %_gep1 = getelementptr [10 x i32], [10 x i32]* %myarr, i32 0, i32 8
  store i32 120, i32* %_gep1, align 4
  %_gep2 = getelementptr [10 x i32], [10 x i32]* %myarr, i32 0, i32 8
  %gep = load i32, i32* %_gep2, align 4
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.2, i32 0, i32 0), i32 %gep)
  %myarr3 = load [10 x i32], [10 x i32]* %myarr, align 4
  ret [10 x i32] %myarr3
}

define [10 x i32] @passarr([10 x i32] %arr10) {
entry:
  %arr101 = alloca [10 x i32], align 4
  store [10 x i32] %arr10, [10 x i32]* %arr101, align 4
  %arr102 = load [10 x i32], [10 x i32]* %arr101, align 4
  ret [10 x i32] %arr102
}
```

Arrays with SDL:

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

```
    int y;
    int r;
    int xspeed;
    int yspeed;
    int window_w;
    int window_h;
    int total;
    arr[100] balls;
    window_w = 600;
    window_h = 600;
    r = 10;
    total = 20;

    for(i = 0; i < total*5; i = i+5){
        balls[i] = (i*10)+50;
    }

    for(i = 1; i < total*5; i = i+5){
        balls[i] = (i*5)+50;
    }
    for(i = 2; i < total*5; i = i+5){
        balls[i] = r;
    }
    for(i = 3; i < total*5; i = i+5){
        balls[i] = (i/5)+1;
    }

    for(i = 4; i < total*5; i = i+5){
        balls[i] = (i/5)+1;
    }

    createWindow(window_w, window_h);
    background(255,255,255);
    for (;1<2;) {
        color(0,0,255);
        for(i = 0; i < total*5; i = i + 5){
            x = balls[i];
            y = balls[i+1];
            r = balls[i+2];
            xspeed = balls[i+3];
            yspeed = balls[i+4];
            color(i*10,0,i*10);
            drawCircle(x, y, r);
            if (x > window_w-r) balls[i+3] = -(balls[i+3]);
            if (x < r) balls[i+3] = -(balls[i+3]);
            if (y > window_h-r) balls[i+4] = -(balls[i+4]);
            if (y < r) balls[i+4] = -(balls[i+4]);

            balls[i] = x + balls[i+3];
            balls[i+1] = y + balls[i+4];

        }
        draw();

    }
    return 0;
}
```

```
; ModuleID = 'compArt'
source_filename = "compArt"

@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

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

declare i32 @printbig(i32)

declare i32 @draw()

declare i32 @createWindow(i32, i32)

declare i32 @background(i32, i32, i32)

declare i32 @color(i32, i32, i32)

declare i32 @opacity(i32)

declare i32 @fill()

declare i32 @noFill()

declare i32 @drawRect(i32, i32, i32, i32)

declare i32 @drawLine(i32, i32, i32, i32)

declare i32 @drawCircle(i32, i32, i32)

declare i32 @drawTriangle(i32, i32, i32, i32, i32, i32)

declare i32 @getMouseX()

declare i32 @getMouseY()

define i32 @main() {
entry:
  %i = alloca i32, align 4
  %x = alloca i32, align 4
  %y = alloca i32, align 4
  %r = alloca i32, align 4
  %xspeed = alloca i32, align 4
  %yspeed = alloca i32, align 4
  %window_w = alloca i32, align 4
  %window_h = alloca i32, align 4
  %total = alloca i32, align 4
  %balls = alloca [100 x i32], align 4
  store i32 600, i32* %window_w, align 4
  store i32 600, i32* %window_h, align 4
  store i32 10, i32* %r, align 4
  store i32 20, i32* %total, align 4
  store i32 0, i32* %i, align 4
  br label %while

while:                                            ; preds = %
```

```
    while_body, %entry
  %i6 = load i32, i32* %i, align 4
  %total7 = load i32, i32* %total, align 4
  %tmp8 = mul i32 %total7, 5
  %tmp9 = icmp slt i32 %i6, %tmp8
  br i1 %tmp9, label %while_body, label %merge

while_body:                                       ; preds = %while
  %i1 = load i32, i32* %i, align 4
  %i2 = load i32, i32* %i, align 4
  %tmp = mul i32 %i2, 10
  %tmp3 = add i32 %tmp, 50
  %_gep = getelementptr [100 x i32], [100 x i32]* %balls, i32 0,
    i32 %i1
  store i32 %tmp3, i32* %_gep, align 4
  %i4 = load i32, i32* %i, align 4
  %tmp5 = add i32 %i4, 5
  store i32 %tmp5, i32* %i, align 4
  br label %while

merge:                                            ; preds = %while
  store i32 1, i32* %i, align 4
  br label %while10

while10:                                          ; preds = %
    while_body11, %merge
  %i19 = load i32, i32* %i, align 4
  %total20 = load i32, i32* %total, align 4
  %tmp21 = mul i32 %total20, 5
  %tmp22 = icmp slt i32 %i19, %tmp21
  br i1 %tmp22, label %while_body11, label %merge23

while_body11:                                     ; preds = %
    while10
  %i12 = load i32, i32* %i, align 4
  %i13 = load i32, i32* %i, align 4
  %tmp14 = mul i32 %i13, 5
  %tmp15 = add i32 %tmp14, 50
  %_gep16 = getelementptr [100 x i32], [100 x i32]* %balls, i32 0,
    i32 %i12
  store i32 %tmp15, i32* %_gep16, align 4
  %i17 = load i32, i32* %i, align 4
  %tmp18 = add i32 %i17, 5
  store i32 %tmp18, i32* %i, align 4
  br label %while10

merge23:                                          ; preds = %
    while10
  store i32 2, i32* %i, align 4
  br label %while24

while24:                                          ; preds = %
    while_body25, %merge23
  %i31 = load i32, i32* %i, align 4
  %total32 = load i32, i32* %total, align 4
  %tmp33 = mul i32 %total32, 5
  %tmp34 = icmp slt i32 %i31, %tmp33
```

```
107   br i1 %tmp34, label %while_body25, label %merge35
108
109 while_body25:                                     ; preds = %
      while24
110   %i26 = load i32, i32* %i, align 4
111   %r27 = load i32, i32* %r, align 4
112   %_gep28 = getelementptr [100 x i32], [100 x i32]* %balls, i32 0,
        i32 %i26
113   store i32 %r27, i32* %_gep28, align 4
114   %i29 = load i32, i32* %i, align 4
115   %tmp30 = add i32 %i29, 5
116   store i32 %tmp30, i32* %i, align 4
117   br label %while24
118
119 merge35:                                          ; preds = %
      while24
120   store i32 3, i32* %i, align 4
121   br label %while36
122
123 while36:                                          ; preds = %
      while_body37, %merge35
124   %i45 = load i32, i32* %i, align 4
125   %total46 = load i32, i32* %total, align 4
126   %tmp47 = mul i32 %total46, 5
127   %tmp48 = icmp slt i32 %i45, %tmp47
128   br i1 %tmp48, label %while_body37, label %merge49
129
130 while_body37:                                     ; preds = %
      while36
131   %i38 = load i32, i32* %i, align 4
132   %i39 = load i32, i32* %i, align 4
133   %tmp40 = sdiv i32 %i39, 5
134   %tmp41 = add i32 %tmp40, 1
135   %_gep42 = getelementptr [100 x i32], [100 x i32]* %balls, i32 0,
        i32 %i38
136   store i32 %tmp41, i32* %_gep42, align 4
137   %i43 = load i32, i32* %i, align 4
138   %tmp44 = add i32 %i43, 5
139   store i32 %tmp44, i32* %i, align 4
140   br label %while36
141
142 merge49:                                          ; preds = %
      while36
143   store i32 4, i32* %i, align 4
144   br label %while50
145
146 while50:                                          ; preds = %
      while_body51, %merge49
147   %i59 = load i32, i32* %i, align 4
148   %total60 = load i32, i32* %total, align 4
149   %tmp61 = mul i32 %total60, 5
150   %tmp62 = icmp slt i32 %i59, %tmp61
151   br i1 %tmp62, label %while_body51, label %merge63
152
153 while_body51:                                     ; preds = %
      while50
154   %i52 = load i32, i32* %i, align 4
```

```
155   %i53 = load i32, i32* %i, align 4
156   %tmp54 = sdiv i32 %i53, 5
157   %tmp55 = add i32 %tmp54, 1
158   %_gep56 = getelementptr [100 x i32], [100 x i32]* %balls, i32 0,
        i32 %i52
159   store i32 %tmp55, i32* %_gep56, align 4
160   %i57 = load i32, i32* %i, align 4
161   %tmp58 = add i32 %i57, 5
162   store i32 %tmp58, i32* %i, align 4
163   br label %while50
164
165 merge63:                                          ; preds = %
        while50
166   %window_h64 = load i32, i32* %window_h, align 4
167   %window_w65 = load i32, i32* %window_w, align 4
168   %createWindow = call i32 @createWindow(i32 %window_w65, i32 %
        window_h64)
169   %background = call i32 @background(i32 255, i32 255, i32 255)
170   br label %while66
171
172 while66:                                          ; preds = %
        merge177, %merge63
173   br i1 true, label %while_body67, label %merge178
174
175 while_body67:                                     ; preds = %
        while66
176   %color = call i32 @color(i32 0, i32 0, i32 255)
177   store i32 0, i32* %i, align 4
178   br label %while68
179
180 while68:                                          ; preds = %
        merge143, %while_body67
181   %i173 = load i32, i32* %i, align 4
182   %total174 = load i32, i32* %total, align 4
183   %tmp175 = mul i32 %total174, 5
184   %tmp176 = icmp slt i32 %i173, %tmp175
185   br i1 %tmp176, label %while_body69, label %merge177
186
187 while_body69:                                     ; preds = %
        while68
188   %i70 = load i32, i32* %i, align 4
189   %_gep71 = getelementptr [100 x i32], [100 x i32]* %balls, i32 0,
        i32 %i70
190   %gep = load i32, i32* %_gep71, align 4
191   store i32 %gep, i32* %x, align 4
192   %i72 = load i32, i32* %i, align 4
193   %tmp73 = add i32 %i72, 1
194   %_gep74 = getelementptr [100 x i32], [100 x i32]* %balls, i32 0,
        i32 %tmp73
195   %gep75 = load i32, i32* %_gep74, align 4
196   store i32 %gep75, i32* %y, align 4
197   %i76 = load i32, i32* %i, align 4
198   %tmp77 = add i32 %i76, 2
199   %_gep78 = getelementptr [100 x i32], [100 x i32]* %balls, i32 0,
        i32 %tmp77
200   %gep79 = load i32, i32* %_gep78, align 4
201   store i32 %gep79, i32* %r, align 4
```

```
  %i80 = load i32, i32* %i, align 4
  %tmp81 = add i32 %i80, 3
  %_gep82 = getelementptr [100 x i32], [100 x i32]* %balls, i32 0, i32 %tmp81
  %gep83 = load i32, i32* %_gep82, align 4
  store i32 %gep83, i32* %xspeed, align 4
  %i84 = load i32, i32* %i, align 4
  %tmp85 = add i32 %i84, 4
  %_gep86 = getelementptr [100 x i32], [100 x i32]* %balls, i32 0, i32 %tmp85
  %gep87 = load i32, i32* %_gep86, align 4
  store i32 %gep87, i32* %yspeed, align 4
  %i88 = load i32, i32* %i, align 4
  %tmp89 = mul i32 %i88, 10
  %i90 = load i32, i32* %i, align 4
  %tmp91 = mul i32 %i90, 10
  %color92 = call i32 @color(i32 %tmp91, i32 0, i32 %tmp89)
  %r93 = load i32, i32* %r, align 4
  %y94 = load i32, i32* %y, align 4
  %x95 = load i32, i32* %x, align 4
  %drawCircle = call i32 @drawCircle(i32 %x95, i32 %y94, i32 %r93)
  %x96 = load i32, i32* %x, align 4
  %window_w97 = load i32, i32* %window_w, align 4
  %r98 = load i32, i32* %r, align 4
  %tmp99 = sub i32 %window_w97, %r98
  %tmp100 = icmp sgt i32 %x96, %tmp99
  br i1 %tmp100, label %then, label %else

merge101:                                         ; preds = %else, %then
  %x110 = load i32, i32* %x, align 4
  %r111 = load i32, i32* %r, align 4
  %tmp112 = icmp slt i32 %x110, %r111
  br i1 %tmp112, label %then114, label %else123

then:                                             ; preds = %while_body69
  %i102 = load i32, i32* %i, align 4
  %tmp103 = add i32 %i102, 3
  %i104 = load i32, i32* %i, align 4
  %tmp105 = add i32 %i104, 3
  %_gep106 = getelementptr [100 x i32], [100 x i32]* %balls, i32 0, i32 %tmp105
  %gep107 = load i32, i32* %_gep106, align 4
  %tmp108 = sub i32 0, %gep107
  %_gep109 = getelementptr [100 x i32], [100 x i32]* %balls, i32 0, i32 %tmp103
  store i32 %tmp108, i32* %_gep109, align 4
  br label %merge101

else:                                             ; preds = %while_body69
  br label %merge101

merge113:                                         ; preds = %else123, %then114
  %y124 = load i32, i32* %y, align 4
```

```
  %window_h125 = load i32, i32* %window_h, align 4
  %r126 = load i32, i32* %r, align 4
  %tmp127 = sub i32 %window_h125, %r126
  %tmp128 = icmp sgt i32 %y124, %tmp127
  br i1 %tmp128, label %then130, label %else139

then114:                                          ; preds = %
    merge101
  %i115 = load i32, i32* %i, align 4
  %tmp116 = add i32 %i115, 3
  %i117 = load i32, i32* %i, align 4
  %tmp118 = add i32 %i117, 3
  %_gep119 = getelementptr [100 x i32], [100 x i32]* %balls, i32 0,
     i32 %tmp118
  %gep120 = load i32, i32* %_gep119, align 4
  %tmp121 = sub i32 0, %gep120
  %_gep122 = getelementptr [100 x i32], [100 x i32]* %balls, i32 0,
     i32 %tmp116
  store i32 %tmp121, i32* %_gep122, align 4
  br label %merge113

else123:                                          ; preds = %
    merge101
  br label %merge113

merge129:                                         ; preds = %
    else139, %then130
  %y140 = load i32, i32* %y, align 4
  %r141 = load i32, i32* %r, align 4
  %tmp142 = icmp slt i32 %y140, %r141
  br i1 %tmp142, label %then144, label %else153

then130:                                          ; preds = %
    merge113
  %i131 = load i32, i32* %i, align 4
  %tmp132 = add i32 %i131, 4
  %i133 = load i32, i32* %i, align 4
  %tmp134 = add i32 %i133, 4
  %_gep135 = getelementptr [100 x i32], [100 x i32]* %balls, i32 0,
     i32 %tmp134
  %gep136 = load i32, i32* %_gep135, align 4
  %tmp137 = sub i32 0, %gep136
  %_gep138 = getelementptr [100 x i32], [100 x i32]* %balls, i32 0,
     i32 %tmp132
  store i32 %tmp137, i32* %_gep138, align 4
  br label %merge129

else139:                                          ; preds = %
    merge113
  br label %merge129

merge143:                                         ; preds = %
    else153, %then144
  %i154 = load i32, i32* %i, align 4
  %x155 = load i32, i32* %x, align 4
  %i156 = load i32, i32* %i, align 4
  %tmp157 = add i32 %i156, 3
```

```
  %_gep158 = getelementptr [100 x i32], [100 x i32]* %balls, i32 0, i32 %tmp157
  %gep159 = load i32, i32* %_gep158, align 4
  %tmp160 = add i32 %x155, %gep159
  %_gep161 = getelementptr [100 x i32], [100 x i32]* %balls, i32 0, i32 %i154
  store i32 %tmp160, i32* %_gep161, align 4
  %i162 = load i32, i32* %i, align 4
  %tmp163 = add i32 %i162, 1
  %y164 = load i32, i32* %y, align 4
  %i165 = load i32, i32* %i, align 4
  %tmp166 = add i32 %i165, 4
  %_gep167 = getelementptr [100 x i32], [100 x i32]* %balls, i32 0, i32 %tmp166
  %gep168 = load i32, i32* %_gep167, align 4
  %tmp169 = add i32 %y164, %gep168
  %_gep170 = getelementptr [100 x i32], [100 x i32]* %balls, i32 0, i32 %tmp163
  store i32 %tmp169, i32* %_gep170, align 4
  %i171 = load i32, i32* %i, align 4
  %tmp172 = add i32 %i171, 5
  store i32 %tmp172, i32* %i, align 4
  br label %while68

then144:                                          ; preds = %merge129
  %i145 = load i32, i32* %i, align 4
  %tmp146 = add i32 %i145, 4
  %i147 = load i32, i32* %i, align 4
  %tmp148 = add i32 %i147, 4
  %_gep149 = getelementptr [100 x i32], [100 x i32]* %balls, i32 0, i32 %tmp148
  %gep150 = load i32, i32* %_gep149, align 4
  %tmp151 = sub i32 0, %gep150
  %_gep152 = getelementptr [100 x i32], [100 x i32]* %balls, i32 0, i32 %tmp146
  store i32 %tmp151, i32* %_gep152, align 4
  br label %merge143

else153:                                          ; preds = %merge129
  br label %merge143

merge177:                                         ; preds = %while68
  %draw = call i32 @draw()
  br label %while66

merge178:                                         ; preds = %while66
  ret i32 0
}
```

7 Lessons Learned

1. Aaron Priven:
   (a) One of the critical lessons I learned from this project was the importance of testing code often. There were times where we made the mistake of making several changes without testing them, and it ended up being a mess to debug. As the semester progressed, we were much more careful to constantly test any updates to ensure that every change was not the source of any bug.
   (b) I also learned the importance of creating automated testing and running systems. When compiling the same test case many times, it significantly speeds up the workflow to create bash scripts to handle all of the compilation and execution, instead of taking the time to write out each command manually.
2. Julia Reichel:
   (a) I learned the struggle and later satisfaction of linking external libraries into our code. We struggled to link the SDL library with our compiler. Once we became more familiar with how to create built-in-functions in the Codegen, we were able to leverage a helper file that called upon the SDL library functions.
   (b) Additionally, I would like to stress the importance of testing out every single feature that is added and making sure that your files compile every step of the way. Although this is a slow process, it will greatly help one find bugs in the code and progress forward.
   (c) I learned the importance of communication. Our team made sure to update one another frequently on what we each were working on and utilized our group chat tremendously. Whenever one of our team members got stuck, we offered to hop on a call and work together to figure out the issue at hand.
3. Asher Willner:
   (a) I learned how important it is to ask for help when stuck. When implementing arrays, Evan and I found ourselves constantly stuck and confused on how to move forward with the LLVM Ocaml API. We went back and forth with our TA Harry, who was super helpful in guiding us through the challenges.
   (b) I also learned that I need to be able to cope with failure and setbacks. Evan and I spent many hours trying to figure out 5 lines of code, only in the end to realize it doesn't really help us and we need to roll back. Keeping a positive mindset and making incremental improvements proved to be key.

(c) Lastly, I learned that divide and conquer is key to accomplishing big tasks. Without my amazing teammates and our ability to work together while also separately, we wouldn't have been able to put both sides together, implementing arrays and the drawing of animations.

4. Evan Zauderer:

   (a) In this project I really learned just how important teammates are. This project is extremely daunting, and at times the task seems impossible to compete. However, while working together to edge closer and closer to the final product, the project becomes enjoyable and highly rewarding. I feel like we all have a great sense of pride looking back on what we made together in just one semester.

   (b) I also learned the importance of keeping to a schedule on this project. There are many steps to this project, and if the group falls behind the work will pile up at the end of the semester, when everyone is already busy. Therefore, it couldn't be more important to space out the work so that only the finishing touches are left for the end, when everyone is scrapping for minutes of time to work together.

   (c) Perseverance is key. Every once in a while an issue would arise that would seem impossible to solve. However, staying diligent and keeping up the hard work is the best way to get through the struggle, as we clearly learned.

8 Appendix

This section contains all of the files we have worked on throughout the semester. Note that the authors of each program is explained above in section 5.2. The split was very even.

8.1 Compiler

compArtParse.mly:

```
/* Ocamlyacc parser for CompArt */

%{
open Ast
%}

%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA PLUS MINUS TIMES
    DIVIDE ASSIGN
%token NOT EQ NEQ LT LEQ GT GEQ AND OR RBRACKET LBRACKET ARRAY
%token RETURN IF ELSE FOR WHILE INT BOOL FLOAT VOID
%token <int> LITERAL
%token <bool> BLIT
%token <string> ID FLIT
%token EOF

%start program
%type <Ast.program> program

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

%%

program:
  decls EOF { $1 }

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

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

```
45    body = List.rev $8 } }
46
47 formals_opt:
48     /* nothing */ { [] }
49   | formal_list   { $1 }
50
51 formal_list:
52     typ ID                   { [($1,$2)]     }
53   | formal_list COMMA typ ID { ($3,$4) :: $1 }
54
55 typ:
56     INT   { Int   }
57   | BOOL  { Bool  }
58   | FLOAT { Float }
59   | VOID  { Void  }
60   | ARRAY LBRACKET LITERAL RBRACKET { Array($3) }
61
62 vdecl_list:
63     /* nothing */    { [] }
64   | vdecl_list vdecl { $2 :: $1 }
65
66 vdecl:
67    typ ID SEMI { ($1, $2) }
68
69 stmt_list:
70     /* nothing */  { [] }
71   | stmt_list stmt { $2 :: $1 }
72
73 stmt:
74     expr SEMI                                { Expr $1
       }
75   | RETURN expr_opt SEMI                     { Return $2
       }
76   | LBRACE stmt_list RBRACE                  { Block(List.rev $2)
       }
77   | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([]))
       }
78   | IF LPAREN expr RPAREN stmt ELSE stmt     { If($3, $5, $7)
       }
79   | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
80                                              { For($3, $5, $7, $9)
       }
81   | WHILE LPAREN expr RPAREN stmt            { While($3, $5)
       }
82
83 expr_opt:
84     /* nothing */ { Noexpr }
85   | expr          { $1 }
86
87 expr:
88     LITERAL          { Literal($1)            }
89   | FLIT       { Fliteral($1)           }
90   | BLIT             { BoolLit($1)            }
91   | ID               { Id($1)                 }
92   | expr PLUS   expr { Binop($1, Add,   $3)   }
93   | expr MINUS  expr { Binop($1, Sub,   $3)   }
94   | expr TIMES  expr { Binop($1, Mult,  $3)   }
```

```
  | expr DIVIDE expr { Binop($1, Div,   $3)   }
  | expr EQ     expr { Binop($1, Equal, $3)   }
  | expr NEQ    expr { Binop($1, Neq,   $3)   }
  | expr LT     expr { Binop($1, Less,  $3)   }
  | expr LEQ    expr { Binop($1, Leq,   $3)   }
  | expr GT     expr { Binop($1, Greater, $3) }
  | expr GEQ    expr { Binop($1, Geq,   $3)   }
  | expr AND    expr { Binop($1, And,   $3)   }
  | expr OR     expr { Binop($1, Or,    $3)   }
  | MINUS expr %prec NOT { Unop(Neg, $2)      }
  | NOT expr         { Unop(Not, $2)          }
  | ID ASSIGN expr   { Assign($1, $3)         }
  | ID LPAREN args_opt RPAREN { Call($1, $3)  }
  | LPAREN expr RPAREN { $2                   }
  | ID LBRACKET expr RBRACKET ASSIGN expr { ArrAssign($1, $3, $6) }
  | ID LBRACKET expr RBRACKET { ArrAccess($1, $3) }

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

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

scanner.mll:

```
(* Ocamllex scanner for CompArt *)

{ open CompArtParse }

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

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

```
| "!"      { NOT }
| "if"     { IF }
| "else"   { ELSE }
| "for"    { FOR }
| "while"  { WHILE }
| "return" { RETURN }
| "int"    { INT }
| "bool"   { BOOL }
| "float"  { FLOAT }
| "void"   { VOID }
| "true"   { BLIT(true)  }
| "false"  { BLIT(false) }
| "arr"    { ARRAY }
| digits as lxm { LITERAL(int_of_string lxm) }
| digits '.'  digit* ( ['e' 'E'] ['+' '-']? digits )? as lxm { FLIT
    (lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*     as lxm { ID(
    lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped
    char)) }

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

ast.ml:

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

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

type uop = Neg | Not

type typ = Int | Bool | Float | Void | Array of int

type bind = typ * string

type expr =
    Literal of int
  | Fliteral of string
  | BoolLit of bool
  | Id of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Assign of string * expr
  | Call of string * expr list
  | Noexpr
  | ArrAssign of string * expr * expr
  | ArrAccess of string * expr
  (* | Array of string * int *)

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

type program = bind list * func_decl list

(* Pretty-printing functions *)

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

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

let rec string_of_expr = function
    Literal(l) -> string_of_int l
  | Fliteral(l) -> l
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | Id(s) -> s
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^
    string_of_expr e2
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
  | Assign(v, e) -> v ^ " = " ^ string_of_expr e
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^
    ")"
  | Noexpr -> ""
  | ArrAssign(s, e1, e2) -> s ^ "[" ^ string_of_expr e1 ^ "] = " ^
    string_of_expr e2
  | ArrAccess(s, e)  -> s ^ "[" ^ string_of_expr e ^ "]"
  (* | Array(s, i) -> s ^ " [" ^ string_of_int i ^ "]" *)

let rec string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^
```

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

let string_of_typ = function
    Int -> "int"
  | Bool -> "bool"
  | Float -> "float"
  | Void -> "void"
  | Array(s) -> "array of size " ^ string_of_int s

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

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

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

sast.ml:

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

open Ast

type sexpr = typ * sx
and sx =
    SLiteral of int
  | SFliteral of string
  | SBoolLit of bool
  | SId of string
  | SBinop of sexpr * op * sexpr
  | SUnop of uop * sexpr
  | SAssign of string * sexpr
  | SCall of string * sexpr list
  | SNoexpr
  | SArrAssign of string * sexpr * sexpr
  | SArrAccess of string * sexpr

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

type sprogram = bind list * sfunc_decl list

(* Pretty-printing functions *)

let rec string_of_sexpr (t, e) =
  "(" ^ string_of_typ t ^ " : " ^ (match e with
    SLiteral(l) -> string_of_int l
  | SBoolLit(true) -> "true"
  | SBoolLit(false) -> "false"
  | SFliteral(l) -> l
  | SId(s) -> s
  | SBinop(e1, o, e2) ->
      string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^
    string_of_sexpr e2
  | SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
  | SAssign(v, e) -> v ^ " = " ^ string_of_sexpr e
  | SCall(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^
    ")"
  | SNoexpr -> ""
  | SArrAssign(s, e1, e2) -> s ^ "[" ^ string_of_sexpr e1 ^ "] = "
    ^ string_of_sexpr e2
  | SArrAccess(s, e) -> s ^ "[" ^ string_of_sexpr e ^ "]"
        ) ^ ")"

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

```

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

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

semant.ml:

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

open Ast
open Sast

module StringMap = Map.Make(String)

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

   Check each global variable, then check each function *)

let check (globals, functions) =

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

  let check_binds_globals (kind : string) (binds : bind list) =
    List.iter (function
  (Void, b) -> raise (Failure ("illegal void " ^ kind ^ " " ^ b))
  | (Array(_), b) -> raise (Failure ("illegal array in global
    context " ^ kind ^ " " ^ b))
      | _ -> ()) binds;
    let rec dups = function
        [] -> ()
      | ((_,n1) :: (_,n2) :: _) when n1 = n2 ->
    raise (Failure ("duplicate " ^ kind ^ " " ^ n1))
      | _ :: t -> dups t
    in dups (List.sort (fun (_,a) (_,b) -> compare a b) binds)
  in

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

```
42
43  check_binds_globals "global" globals;
44
45  (**** Check functions ****)
46
47  (* Collect function declarations for built-in functions: no
      bodies *)
48  let built_in_decls =
49    let add_bind map (name, binds) = StringMap.add name {
50      typ = Int;
51      fname = name;
52      formals = binds;
53      locals = []; body = [] } map
54    in List.fold_left add_bind StringMap.empty [ ("print", [(Int, "
      x")]);
55                                 ("printb", [(Bool, "x")]);
56                                 ("printf", [(Float, "x")]);
57                                 ("printbig", [(Int, "x")]);
58                                 ("draw", [ ]);
59                                 ("createWindow", [(Int, "w"); (Int,
      "h")]);
60                                 ("background", [(Int, "r"); (Int, "g
      "); (Int, "b")]);
61                                 ("color", [(Int, "r"); (Int, "g"); (
      Int, "b")]);
62                                 ("opacity", [(Int, "x")]);
63                                 ("fill", [ ]);
64                                 ("noFill", [ ]);
65                                 ("drawRect", [(Int, "x1"); (Int, "y1
      "); (Int, "x2"); (Int, "y2")]);
66                                 ("drawCircle", [(Int, "x"); (Int, "y
      "); (Int, "r")]);
67                                 ("drawTriangle", [(Int, "x1"); (Int,
       "y1"); (Int, "x2"); (Int, "y2");(Int, "x3"); (Int, "y3")]);
68                                 ("getMouseX", [ ]);
69                                 ("getMouseY", [ ]);
70                                 ("drawLine", [(Int, "x1"); (Int, "y1
      "); (Int, "x2"); (Int, "y2")])]
71  in
72
73  (* Add function name to symbol table *)
74  let add_func map fd =
75    let built_in_err = "function " ^ fd.fname ^ " may not be
      defined"
76    and dup_err = "duplicate function " ^ fd.fname
77    and make_err er = raise (Failure er)
78    and n = fd.fname (* Name of the function *)
79    in match fd with (* No duplicate functions or redefinitions of
      built-ins *)
80         _ when StringMap.mem n built_in_decls -> make_err
      built_in_err
81       | _ when StringMap.mem n map -> make_err dup_err
82       | _ ->  StringMap.add n fd map
83  in
84
85  (* Collect all function names into one symbol table *)
86  let function_decls = List.fold_left add_func built_in_decls
```

```
    functions
  in

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

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

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

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

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

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

    (* Return a semantically-checked expression, i.e., with a type
    *)
    let rec expr = function
        Literal  l -> (Int, SLiteral l)
      | Fliteral l -> (Float, SFliteral l)
      | BoolLit l  -> (Bool, SBoolLit l)
      | Noexpr     -> (Void, SNoexpr)
      | Id s       -> (type_of_identifier s, SId s)
      | Assign(var, e) as ex ->
          let lt = type_of_identifier var
          and (rt, e') = expr e in
          let err = "illegal assignment " ^ string_of_typ lt ^ " =
    " ^
            string_of_typ rt ^ " in " ^ string_of_expr ex
          in (check_assign lt rt err, SAssign(var, (rt, e')))

      | ArrAssign(s, e1, e2) as ex -> let (rt1, e1') = expr e1 and
    (rt2, e2') = expr e2 in
                let err2 =
```

```
135               "illegal assignment of " ^ string_of_typ rt2 ^ " in
     " ^ string_of_expr ex
136               in (check_assign Int rt2 err2,
137                   SArrAssign(s, (rt1, e1'), (rt2, e2'')))
138
139       | ArrAccess(s, e) as ex -> let lt = type_of_identifier s
140               and (rt, e') = expr e in
141               let err = "illegal assignment " ^ string_of_typ lt
    ^ " = " ^
142                 string_of_typ rt ^ " in " ^ string_of_expr ex
143               in (check_assign Int rt err, SArrAccess(s, (rt, e')
    ))
144
145       | Unop(op, e) as ex ->
146           let (t, e') = expr e in
147           let ty = match op with
148             Neg when t = Int || t = Float -> t
149           | Not when t = Bool -> Bool
150           | _ -> raise (Failure ("illegal unary operator " ^
151                                  string_of_uop op ^ string_of_typ t
     ^
152                                  " in " ^ string_of_expr ex))
153           in (ty, SUnop(op, (t, e')))
154       | Binop(e1, op, e2) as e ->
155           let (t1, e1') = expr e1
156           and (t2, e2') = expr e2 in
157           (* All binary operators require operands of the same type
     *)
158           let same = t1 = t2 in
159           (* Determine expression type based on operator and
    operand types *)
160           let ty = match op with
161             Add | Sub | Mult | Div when same && t1 = Int   -> Int
162           | Add | Sub | Mult | Div when same && t1 = Float -> Float
163           | Equal | Neq            when same               -> Bool
164           | Less | Leq | Greater | Geq
165                      when same && (t1 = Int || t1 = Float) -> Bool
166           | And | Or when same && t1 = Bool -> Bool
167           | _ -> raise (
168         Failure ("illegal binary operator " ^
169                         string_of_typ t1 ^ " " ^ string_of_op op ^ "
     " ^
170                         string_of_typ t2 ^ " in " ^ string_of_expr e
    ))
171           in (ty, SBinop((t1, e1'), op, (t2, e2')))
172       | Call(fname, args) as call ->
173           let fd = find_func fname in
174           let param_length = List.length fd.formals in
175           if List.length args != param_length then
176             raise (Failure ("expecting " ^ string_of_int
    param_length ^
177                             " arguments in " ^ string_of_expr call)
    )
178           else let check_call (ft, _) e =
179             let (et, e') = expr e in
180             let err = "illegal argument found " ^ string_of_typ et
    ^
```

```
181               " expected " ^ string_of_typ ft ^ " in " ^
    string_of_expr e
182             in (check_assign ft et err, e')
183           in
184           let args' = List.map2 check_call fd.formals args
185           in (fd.typ, SCall(fname, args'))
186     in
187
188     let check_bool_expr e =
189       let (t', e') = expr e
190       and err = "expected Boolean expression in " ^ string_of_expr
    e
191       in if t' != Bool then raise (Failure err) else (t', e')
192     in
193
194     (* Return a semantically-checked statement i.e. containing
    sexprs *)
195     let rec check_stmt = function
196         Expr e -> SExpr (expr e)
197       | If(p, b1, b2) -> SIf(check_bool_expr p, check_stmt b1,
    check_stmt b2)
198       | For(e1, e2, e3, st) ->
199     SFor(expr e1, check_bool_expr e2, expr e3, check_stmt st)
200       | While(p, s) -> SWhile(check_bool_expr p, check_stmt s)
201       | Return e -> let (t, e') = expr e in
202         if t = func.typ then SReturn (t, e')
203         else raise (
204     Failure ("return gives " ^ string_of_typ t ^ " expected " ^
205        string_of_typ func.typ ^ " in " ^ string_of_expr e))
206
207       (* A block is correct if each statement is correct and
    nothing
208          follows any Return statement.  Nested blocks are flattened
    . *)
209       | Block sl ->
210           let rec check_stmt_list = function
211               [Return _ as s] -> [check_stmt s]
212             | Return _ :: _   -> raise (Failure "nothing may follow
     a return")
213             | Block sl :: ss  -> check_stmt_list (sl @ ss) (*
    Flatten blocks *)
214             | s :: ss         -> check_stmt s :: check_stmt_list ss
215             | []              -> []
216           in SBlock(check_stmt_list sl)
217
218     in (* body of check_function *)
219     { styp = func.typ;
220       sfname = func.fname;
221       sformals = func.formals;
222       slocals  = func.locals;
223       sbody = match check_stmt (Block func.body) with
224   SBlock(sl) -> sl
225       | _ -> raise (Failure ("internal error: block didn't become a
     block?"))
226     }
227   in (globals, List.map check_function functions)
```

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 "compArt" 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 void_t     = L.void_type   context
  and array_t n  = L.array_type (L.i32_type context) n in

  (* Return the LLVM type for a compArt type *)
  let ltype_of_typ = function
      A.Int   -> i32_t
    | A.Bool  -> i1_t
    | A.Float -> float_t
    | A.Void  -> void_t
    | A.Array(n) -> array_t n
  in

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

  let printf_t : L.lltype =
      L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
  let printf_func : L.llvalue =
```

```
    L.declare_function "printf" printf_t the_module in

let printbig_t : L.lltype =
    L.function_type i32_t [| i32_t |] in
let printbig_func : L.llvalue =
    L.declare_function "printbig" printbig_t the_module in

let draw_t : L.lltype =
    L.function_type i32_t [| |] in
let draw_func : L.llvalue =
    L.declare_function "draw" draw_t the_module in

let createWindow_t : L.lltype =
    L.function_type i32_t [| i32_t; i32_t |] in
let createWindow_func : L.llvalue =
    L.declare_function "createWindow" createWindow_t the_module
  in

let background_t : L.lltype =
    L.function_type i32_t [| i32_t; i32_t; i32_t |] in
let background_func : L.llvalue =
    L.declare_function "background" background_t the_module in

let color_t : L.lltype =
    L.function_type i32_t [| i32_t; i32_t; i32_t |] in
let color_func : L.llvalue =
    L.declare_function "color" color_t the_module in

let opacity_t : L.lltype =
    L.function_type i32_t [| i32_t |] in
let opacity_func : L.llvalue =
    L.declare_function "opacity" opacity_t the_module in

let fill_t : L.lltype =
    L.function_type i32_t [| |] in
let fill_func : L.llvalue =
    L.declare_function "fill" fill_t the_module in

let noFill_t : L.lltype =
    L.function_type i32_t [| |] in
let noFill_func : L.llvalue =
    L.declare_function "noFill" noFill_t the_module in

let drawRect_t : L.lltype =
    L.function_type i32_t [| i32_t; i32_t; i32_t; i32_t |] in
let drawRect_func : L.llvalue =
    L.declare_function "drawRect" drawRect_t the_module in

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

let drawCircle_t : L.lltype =
    L.function_type i32_t [| i32_t; i32_t; i32_t |] in
let drawCircle_func : L.llvalue =
    L.declare_function "drawCircle" drawCircle_t the_module in
```

```
114
115 let drawTriangle_t : L.lltype =
116     L.function_type i32_t [| i32_t; i32_t; i32_t; i32_t; i32_t;
      i32_t |] in
117 let drawTriangle_func : L.llvalue =
118     L.declare_function "drawTriangle" drawTriangle_t the_module
      in
119
120 let getMouseX_t : L.lltype =
121     L.function_type i32_t [| |] in
122 let getMouseX_func : L.llvalue =
123     L.declare_function "getMouseX" getMouseX_t the_module in
124
125 let getMouseY_t : L.lltype =
126     L.function_type i32_t [| |] in
127 let getMouseY_func : L.llvalue =
128     L.declare_function "getMouseY" getMouseY_t the_module in
129
130 (* Define each function (arguments and return type) so we can
131    call it even before we've created its body *)
132 let function_decls : (L.llvalue * sfunc_decl) StringMap.t =
133   let function_decl m fdecl =
134     let name = fdecl.sfname
135     and formal_types =
136 Array.of_list (List.map (fun (t,_) -> ltype_of_typ t) fdecl.
      sformals)
137     in let ftype = L.function_type (ltype_of_typ fdecl.styp)
      formal_types in
138     StringMap.add name (L.define_function name ftype the_module,
      fdecl) m in
139   List.fold_left function_decl StringMap.empty functions in
140
141 (* Fill in the body of the given function *)
142 let build_function_body fdecl =
143   let (the_function, _) = StringMap.find fdecl.sfname
      function_decls in
144   let builder = L.builder_at_end context (L.entry_block
      the_function) in
145
146   let int_format_str = L.build_global_stringptr "%d\n" "fmt"
      builder
147   and float_format_str = L.build_global_stringptr "%g\n" "fmt"
      builder in
148
149   (* Construct the function's "locals": formal arguments and
      locally
150      declared variables.  Allocate each on the stack, initialize
      their
151      value, if appropriate, and remember their values in the "
      locals" map *)
152   let local_vars =
153     let add_formal m (t, n) p =
154       L.set_value_name n p;
155 let local =
156     L.build_alloca (ltype_of_typ t) n builder in
157       ignore (L.build_store p local builder);
158 StringMap.add n local m
```

```
159
160     (* Allocate space for any locally declared variables and add
          the
161        * resulting registers to our map *)
162     and add_local m (t, n) =
163   let local_var =
164     L.build_alloca (ltype_of_typ t) n builder
165   in StringMap.add n local_var m
166     in
167
168     let formals = List.fold_left2 add_formal StringMap.empty
          fdecl.sformals
169         (Array.to_list (L.params the_function)) in
170     List.fold_left add_local formals fdecl.slocals
171   in
172
173   (* Return the value for a variable or formal argument.
174      Check local names first, then global names *)
175   let lookup n = try StringMap.find n local_vars
176                  with Not_found -> StringMap.find n global_vars
177   in
178
179   (* Construct code for an expression; return its value *)
180   let rec expr builder ((_, e) : sexpr) = match e with
181 SLiteral i  -> L.const_int i32_t i
182     | SBoolLit b  -> L.const_int i1_t (if b then 1 else 0)
183     | SFliteral l -> L.const_float_of_string float_t l
184     | SNoexpr     -> L.const_int i32_t 0
185     | SId s       -> L.build_load (lookup s) s builder
186     | SAssign (s, e) -> let e' = expr builder e in
187                         ignore(L.build_store e' (lookup s)
          builder); e'
188
189     | SArrAssign (s, e1, e2) -> let e1' = expr builder e1 and e2'
         = expr builder e2
190                                 and arr = (lookup s) in
191                                 (ignore(L.build_store e2'
192                                 ( Llvm.build_gep arr [| (Llvm.
          const_int i32_t 0);
193
                  (e1') |]
194                                     ("_gep") builder
195                                 )
196                                 builder); e2')
197
198     | SArrAccess (s, e) -> let e' = expr builder e and arr =
          lookup s in
199                            let gep_ptr = Llvm.build_gep arr [| (
          Llvm.const_int i32_t 0);
200
            (e') |]
201                              ("_gep") builder in
202                            let result = Llvm.build_load gep_ptr
          "gep" builder in
203                            result
204
205     | SBinop ((A.Float,_ ) as e1, op, e2) ->
```

```
206     let e1' = expr builder e1
207     and e2' = expr builder e2 in
208     (match op with
209       A.Add     -> L.build_fadd
210     | A.Sub     -> L.build_fsub
211     | A.Mult    -> L.build_fmul
212     | A.Div     -> L.build_fdiv
213     | A.Equal   -> L.build_fcmp L.Fcmp.Oeq
214     | A.Neq     -> L.build_fcmp L.Fcmp.One
215     | A.Less    -> L.build_fcmp L.Fcmp.Olt
216     | A.Leq     -> L.build_fcmp L.Fcmp.Ole
217     | A.Greater -> L.build_fcmp L.Fcmp.Ogt
218     | A.Geq     -> L.build_fcmp L.Fcmp.Oge
219     | A.And | A.Or ->
220         raise (Failure "internal error: semant should have rejected
         and/or on float")
221     ) e1' e2' "tmp" builder
222       | SBinop (e1, op, e2) ->
223     let e1' = expr builder e1
224     and e2' = expr builder e2 in
225     (match op with
226       A.Add     -> L.build_add
227     | A.Sub     -> L.build_sub
228     | A.Mult    -> L.build_mul
229           | A.Div     -> L.build_sdiv
230     | A.And     -> L.build_and
231     | A.Or      -> L.build_or
232     | A.Equal   -> L.build_icmp L.Icmp.Eq
233     | A.Neq     -> L.build_icmp L.Icmp.Ne
234     | A.Less    -> L.build_icmp L.Icmp.Slt
235     | A.Leq     -> L.build_icmp L.Icmp.Sle
236     | A.Greater -> L.build_icmp L.Icmp.Sgt
237     | A.Geq     -> L.build_icmp L.Icmp.Sge
238     ) e1' e2' "tmp" builder
239       | SUnop(op, ((t, _) as e)) ->
240           let e' = expr builder e in
241     (match op with
242       A.Neg when t = A.Float -> L.build_fneg
243     | A.Neg                 -> L.build_neg
244           | A.Not                  -> L.build_not) e' "tmp" builder
245       | SCall ("print", [e]) | SCall ("printb", [e]) ->
246     L.build_call printf_func [| int_format_str ; (expr builder e)
      |]
247       "printf" builder
248       | SCall ("printbig", [e]) ->
249     L.build_call printbig_func [| (expr builder e) |] "printbig"
      builder
250       | SCall ("draw", []) ->
251     L.build_call draw_func [| |] "draw" builder
252       | SCall ("createWindow", [w;h]) ->
253     L.build_call createWindow_func [| (expr builder w);(expr
      builder h) |] "createWindow" builder
254       | SCall ("background", [r;g;b]) ->
255     L.build_call background_func [| (expr builder r);(expr builder
      g);(expr builder b) |] "background" builder
256       | SCall ("color", [r;g;b]) ->
257     L.build_call color_func [| (expr builder r);(expr builder g);(
```

```
    expr builder b) |] "color" builder
258       | SCall ("opacity", [e]) ->
259     L.build_call opacity_func [| (expr builder e)|] "opacity"
    builder
260       | SCall ("fill", []) ->
261     L.build_call fill_func [| |] "fill" builder
262       | SCall ("noFill", []) ->
263     L.build_call noFill_func [| |] "noFill" builder
264       | SCall ("drawRect", [x1;y1;x2;y2]) ->
265     L.build_call drawRect_func [| (expr builder x1); (expr builder
    y1);(expr builder x2);(expr builder y2) |] "drawRect" builder
266       | SCall ("drawLine", [x1;y1;x2;y2]) ->
267     L.build_call drawLine_func [| (expr builder x1); (expr builder
    y1);(expr builder x2);(expr builder y2) |] "drawLine" builder
268       | SCall ("drawCircle", [x;y;r]) ->
269     L.build_call drawCircle_func [| (expr builder x); (expr builder
     y); (expr builder r) |] "drawCircle" builder
270       | SCall ("drawTriangle", [x1;y1;x2;y2;x3;y3]) ->
271     L.build_call drawTriangle_func [| (expr builder x1); (expr
    builder y1);(expr builder x2);(expr builder y2);(expr builder
    x3);(expr builder y3) |] "drawTriangle" builder
272       | SCall ("getMouseX", []) ->
273     L.build_call getMouseX_func [| |] "getMouseX" builder
274       | SCall ("getMouseY", []) ->
275     L.build_call getMouseY_func [| |] "getMouseY" builder
276       | SCall ("printf", [e]) ->
277     L.build_call printf_func [| float_format_str ; (expr builder e)
     |]
278       "printf" builder
279       | SCall (f, args) ->
280          let (fdef, fdecl) = StringMap.find f function_decls in
281   let llargs = List.rev (List.map (expr builder) (List.rev args))
    in
282   let result = (match fdecl.styp with
283                         A.Void -> ""
284                       | _ -> f ^ "_result") in
285         L.build_call fdef (Array.of_list llargs) result builder
286     in
287
288     (* LLVM insists each basic block end with exactly one "
    terminator"
289        instruction that transfers control.  This function runs "
    instr builder"
290        if the current block does not already have a terminator.
    Used,
291        e.g., to handle the "fall off the end of the function" case.
     *)
292     let add_terminal builder instr =
293       match L.block_terminator (L.insertion_block builder) with
294   Some _ -> ()
295       | None -> ignore (instr builder) in
296
297     (* Build the code for the given statement; return the builder
    for
298        the statement's successor (i.e., the next instruction will
    be built
299        after the one generated by this call) *)
```

```
300
301     let rec stmt builder = function
302   SBlock sl -> List.fold_left stmt builder sl
303       | SExpr e -> ignore(expr builder e); builder
304       | SReturn e -> ignore(match fdecl.styp with
305                               (* Special "return nothing" instr *)
306                               A.Void -> L.build_ret_void builder
307                               (* Build return statement *)
308                             | _ -> L.build_ret (expr builder e)
    builder );
309                     builder
310       | SIf (predicate, then_stmt, else_stmt) ->
311          let bool_val = expr builder predicate in
312    let merge_bb = L.append_block context "merge" the_function in
313          let build_br_merge = L.build_br merge_bb in (* partial
    function *)
314
315    let then_bb = L.append_block context "then" the_function in
316    add_terminal (stmt (L.builder_at_end context then_bb) then_stmt)
317      build_br_merge;
318
319    let else_bb = L.append_block context "else" the_function in
320    add_terminal (stmt (L.builder_at_end context else_bb) else_stmt)
321      build_br_merge;
322
323    ignore(L.build_cond_br bool_val then_bb else_bb builder);
324    L.builder_at_end context merge_bb
325
326       | SWhile (predicate, body) ->
327     let pred_bb = L.append_block context "while" the_function in
328     ignore(L.build_br pred_bb builder);
329
330     let body_bb = L.append_block context "while_body" the_function
    in
331     add_terminal (stmt (L.builder_at_end context body_bb) body)
332       (L.build_br pred_bb);
333
334     let pred_builder = L.builder_at_end context pred_bb in
335     let bool_val = expr pred_builder predicate in
336
337     let merge_bb = L.append_block context "merge" the_function in
338     ignore(L.build_cond_br bool_val body_bb merge_bb pred_builder);
339     L.builder_at_end context merge_bb
340
341       (* Implement for loops as while loops *)
342       | SFor (e1, e2, e3, body) -> stmt builder
343       ( SBlock [SExpr e1 ; SWhile (e2, SBlock [body ; SExpr e3]) ]
    )
344     in
345
346     (* Build the code for each statement in the function *)
347     let builder = stmt builder (SBlock fdecl.sbody) in
348
349     (* Add a return if the last block falls off the end *)
350     add_terminal builder (match fdecl.styp with
351         A.Void -> L.build_ret_void
352       | A.Float -> L.build_ret (L.const_float float_t 0.0)
```

```
      | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
  in

  List.iter build_function_body functions;
  the_module
```

8.2 SDL CompArt

compArtHelper.c:

```
/*
   CompArt used the basic structure of this file:

   Copyright (C) 1997-2020 Sam Lantinga <slouken@libsdl.org>

   This software is provided 'as-is', without any express or
    implied
   warranty.  In no event will the authors be held liable for any
    damages
   arising from the use of this software.

   Permission is granted to anyone to use this software for any
    purpose,
   including commercial applications, and to alter it and
    redistribute it
   freely.

   This file is created by : Nitin Jain (nitin.j4@samsung.com)
*/

#include <stdlib.h>
#include <stdio.h>

#ifdef __EMSCRIPTEN__
#include <emscripten/emscripten.h>
#endif

#include "SDL2/SDL.h"
#include "SDL2_gfxPrimitives.h"

SDL_Window *window;
SDL_Renderer *renderer;
SDL_Surface *surface;
int done;
int r_global = 0;
int g_global = 0;
int b_global = 0;
int a_global = 255;
int r_background = 255;
int g_background = 255;
int b_background = 255;
int fill_status = 1;
int width_global;
int height_global;
void color(int red, int green, int blue)
{
```

```
43      r_global = red;
44      g_global = green;
45      b_global = blue;
46  }
47
48  void opacity(int x)
49  {
50      a_global = x;
51  }
52
53  void fill()
54  {
55      fill_status = 1;
56  }
57  void noFill()
58  {
59      fill_status = 0;
60  }
61
62  void background(int red, int green, int blue)
63  {
64
65      SDL_Rect darea;
66      /* Get the Size of drawing surface */
67      SDL_RenderGetViewport(renderer, &darea);
68      SDL_SetRenderDrawColor(renderer, red, green, blue, 0xFF);
69
70      r_background = red;
71      g_background = green;
72      b_background = blue;
73  }
74
75  void drawRect(int x1, int y1, int x2, int y2)
76  {
77      if (fill_status)
78      {
79          boxRGBA(renderer, x1, y1, x2, y2, r_global, g_global,
        b_global, a_global);
80      }
81      else
82      {
83          rectangleRGBA(renderer, x1, y1, x2, y2, r_global, g_global,
         b_global, a_global);
84      }
85  }
86
87  void drawLine(int x1, int y1, int x2, int y2)
88  {
89      lineRGBA(renderer, x1, y1, x2, y2, r_global, g_global, b_global
        , a_global);
90  }
91
92  void drawCircle(int x, int y, int r)
93  {
94      if (fill_status)
95      {
```

```
96          filledCircleRGBA(renderer, x, y, r, r_global, g_global,
    b_global, a_global);
97      }
98      else
99      {
100         circleRGBA(renderer, x, y, r, r_global, g_global, b_global,
     a_global);
101     }
102 }
103
104 void drawTriangle(int x1, int y1, int x2, int y2, int x3, int y3)
105 {
106     if (fill_status)
107     {
108         filledTrigonRGBA(renderer, x1, y1, x2, y2, x3, y3, r_global
    , g_global, b_global, a_global);
109     }
110     else
111     {
112         trigonRGBA(renderer, x1, y1, x2, y2, x3, y3, r_global,
    g_global, b_global, a_global);
113     }
114 }
115
116 int getMouseX()
117 {
118     int mouseX;
119     SDL_GetMouseState(&mouseX, NULL);
120     return mouseX;
121 }
122 int getMouseY()
123 {
124     int mouseY;
125     SDL_GetMouseState(NULL, &mouseY);
126     return mouseY;
127 }
128
129 int draw()
130 {
131     /* Got everything on rendering surface,
132        now Update the drawing image on window screen */
133     SDL_UpdateWindowSurface(window);
134     SDL_SetRenderDrawColor(renderer, r_background, g_background,
    b_background, 0xFF);
135     SDL_RenderClear(renderer);
136
137     SDL_Event e;
138
139     while (SDL_PollEvent(&e))
140     {
141
142         /* Re-create when window has been resized */
143         if ((e.type == SDL_WINDOWEVENT) && (e.window.event ==
    SDL_WINDOWEVENT_SIZE_CHANGED))
144         {
145
146             SDL_DestroyRenderer(renderer);
```

```
147
148             surface = SDL_GetWindowSurface(window);
149             renderer = SDL_CreateSoftwareRenderer(surface);
150             /* Clear the rendering surface with the specified color
     */
151             SDL_SetRenderDrawColor(renderer, 0xFF, 0xFF, 0xFF, 0xFF
    );
152             SDL_RenderClear(renderer);
153         }
154
155         if (e.type == SDL_QUIT)
156         {
157             done = 1;
158             SDL_Quit();
159 #ifdef __EMSCRIPTEN__
160             emscripten_cancel_main_loop();
161 #endif
162             return 0;
163         }
164
165         if ((e.type == SDL_KEYDOWN) && (e.key.keysym.sym ==
    SDLK_ESCAPE))
166         {
167             done = 1;
168             SDL_Quit();
169 #ifdef __EMSCRIPTEN__
170             emscripten_cancel_main_loop();
171 #endif
172             return 0;
173         }
174     }
175
176     return 0;
177 }
178
179 int createWindow(int width, int height)
180 {
181     int width_global = width;
182     int height_global = height;
183     /* Enable standard application logging */
184     SDL_LogSetPriority(SDL_LOG_CATEGORY_APPLICATION,
    SDL_LOG_PRIORITY_INFO);
185
186     /* Initialize SDL */
187     if (SDL_Init(SDL_INIT_VIDEO) != 0)
188     {
189         SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "SDL_Init fail :
     %s\n", SDL_GetError());
190         return 1;
191     }
192
193     /* Create window and renderer for given surface */
194     window = SDL_CreateWindow("CompArt", SDL_WINDOWPOS_UNDEFINED,
    SDL_WINDOWPOS_UNDEFINED, width, height, SDL_WINDOW_RESIZABLE);
195     if (!window)
196     {
```

```
197         SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Window creation
     fail : %s\n", SDL_GetError());
198         return 1;
199     }
200     surface = SDL_GetWindowSurface(window);
201     renderer = SDL_CreateSoftwareRenderer(surface);
202     if (!renderer)
203     {
204         SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Render creation
     for surface fail : %s\n", SDL_GetError());
205         return 1;
206     }
207 
208     /* Clear the rendering surface with the specified color */
209     SDL_SetRenderDrawColor(renderer, 0xFF, 0xFF, 0xFF, 0xFF);
210     SDL_RenderClear(renderer);
211 
212     /* Draw the Image on rendering surface */
213     done = 0;
214 #ifdef __EMSCRIPTEN__
215     emscripten_set_main_loop(loop, 0, 1);
216 #else
217 #endif
218 
219     return 0;
220 }
```

Makefile:

```
1  # "make test" Compiles everything and runs the regression tests
2  
3  .PHONY : test
4  test : all testall.sh
5    ./testall.sh
6  
7  .PHONY : draw
8  draw : all scripts/draw.sh testdrawchessboard.o
9    ./scripts/draw.sh
10 
11 .PHONY : setupdraw
12 setupdraw : all scripts/setupdraw.sh
13   ./scripts/setupdraw.sh
14 
15 .PHONY : color
16 color : all scripts/color.sh
17   ./scripts/color.sh
18 
19 .PHONY : drawline
20 drawline : all scripts/drawline.sh
21   ./scripts/drawline.sh
22 
23 .PHONY : gfxcircle
24 gfxcircle : all scripts/gfxcircle.sh
25   ./scripts/gfxcircle.sh
26 
27 .PHONY : movingball
28 movingball : all scripts/movingball.sh
29   ./scripts/movingball.sh
```

```
30
31 .PHONY : movingball-background
32 movingball-background : all scripts/movingball-background.sh
33   ./scripts/movingball-background.sh
34
35 .PHONY : movingball-color
36 movingball-color : all scripts/movingball-color.sh
37   ./scripts/movingball-color.sh
38
39 .PHONY : movingball-opacity
40 movingball-opacity : all scripts/movingball-opacity.sh
41   ./scripts/movingball-opacity.sh
42
43 .PHONY : mouse
44 mouse : all scripts/mouse.sh
45   ./scripts/mouse.sh
46
47 .PHONY : compArtLogo
48 compArtLogo : all scripts/compArtLogo.sh
49   ./scripts/compArtLogo.sh
50
51 .PHONY : multipleMovingBalls
52 multipleMovingBalls : all scripts/multipleMovingBalls.sh
53   ./scripts/multipleMovingBalls.sh
54
55 # "make all" builds the executable as well as the "printbig"
       library designed
56 # to test linking external code
57
58 .PHONY : all
59 all : compArt.native printbig.o compArtHelper.o
60
61 # "make compArt.native" compiles the compiler
62 #
63 # The _tags file controls the operation of ocamlbuild, e.g., by
       including
64 # packages, enabling warnings
65 #
66 # See https://github.com/ocaml/ocamlbuild/blob/master/manual/manual
       .adoc
67
68 compArt.native :
69   opam config exec -- \
70   ocamlbuild -use-ocamlfind compArt.native
71
72 # "make clean" removes all generated files
73
74 .PHONY : clean
75 clean :
76   ocamlbuild -clean
77   rm -rf testall.log ocamlllvm *.diff *.o
78
79 # Testing the "printbig" example
80
81 printbig : printbig.c
82   cc -o printbig -DBUILD_TEST printbig.c
83
```

```
84
85
86  # Building the tarball
87
88  TESTS = \
89    add1 arith1 arith2 arith3 fib float1 float2 float3 for1 for2
        func1 \
90    func2 func3 func4 func5 func6 func7 func8 func9 gcd2 gcd global1
        \
91    global2 global3 hello if1 if2 if3 if4 if5 if6 local1 local2 ops1
        \
92    ops2 printbig var1 var2 while1 while2
93
94  FAILS = \
95    assign1 assign2 assign3 dead1 dead2 expr1 expr2 expr3 float1
        float2 \
96    for1 for2 for3 for4 for5 func1 func2 func3 func4 func5 func6
        func7 \
97    func8 func9 global1 global2 if1 if2 if3 nomain printbig printb
        print \
98    return1 return2 while1 while2
99
100 TESTFILES = $(TESTS:%=test-%.ca) $(TESTS:%=test-%.out) \
101       $(FAILS:%=fail-%.ca) $(FAILS:%=fail-%.err)
102
103 TARFILES = ast.ml sast.ml codegen.ml Makefile _tags compArt.ml
        compArtParse.mly \
104   README scanner.mll semant.ml testall.sh \
105   printbig.c arcade-font.pbm font2c \
106   Dockerfile \
107   $(TESTFILES:%=tests/%)
108
109 compArt.tar.gz : $(TARFILES)
110   cd .. && tar czf compArt/compArt.tar.gz \
111     $(TARFILES:%=compArt/%)
```

printbig.c:

```
1  /*
2   *  A function illustrating how to link C code to code generated
       from LLVM
3   */
4
5  #include <stdio.h>
6
7  /*
8   * Font information: one byte per row, 8 rows per character
9   * In order, space, 0-9, A-Z
10  */
11 static const char font[] = {
12   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
13   0x1c, 0x3e, 0x61, 0x41, 0x43, 0x3e, 0x1c, 0x00,
14   0x00, 0x40, 0x42, 0x7f, 0x7f, 0x40, 0x40, 0x00,
15   0x62, 0x73, 0x79, 0x59, 0x5d, 0x4f, 0x46, 0x00,
16   0x20, 0x61, 0x49, 0x4d, 0x4f, 0x7b, 0x31, 0x00,
17   0x18, 0x1c, 0x16, 0x13, 0x7f, 0x7f, 0x10, 0x00,
18   0x27, 0x67, 0x45, 0x45, 0x45, 0x7d, 0x38, 0x00,
19   0x3c, 0x7e, 0x4b, 0x49, 0x49, 0x79, 0x30, 0x00,
```

```
  0x03, 0x03, 0x71, 0x79, 0x0d, 0x07, 0x03, 0x00,
  0x36, 0x4f, 0x4d, 0x59, 0x59, 0x76, 0x30, 0x00,
  0x06, 0x4f, 0x49, 0x49, 0x69, 0x3f, 0x1e, 0x00,
  0x7c, 0x7e, 0x13, 0x11, 0x13, 0x7e, 0x7c, 0x00,
  0x7f, 0x7f, 0x49, 0x49, 0x49, 0x7f, 0x36, 0x00,
  0x1c, 0x3e, 0x63, 0x41, 0x41, 0x63, 0x22, 0x00,
  0x7f, 0x7f, 0x41, 0x41, 0x63, 0x3e, 0x1c, 0x00,
  0x00, 0x7f, 0x7f, 0x49, 0x49, 0x49, 0x41, 0x00,
  0x7f, 0x7f, 0x09, 0x09, 0x09, 0x09, 0x01, 0x00,
  0x1c, 0x3e, 0x63, 0x41, 0x49, 0x79, 0x79, 0x00,
  0x7f, 0x7f, 0x08, 0x08, 0x08, 0x7f, 0x7f, 0x00,
  0x00, 0x41, 0x41, 0x7f, 0x7f, 0x41, 0x41, 0x00,
  0x20, 0x60, 0x40, 0x40, 0x40, 0x7f, 0x3f, 0x00,
  0x7f, 0x7f, 0x18, 0x3c, 0x76, 0x63, 0x41, 0x00,
  0x00, 0x7f, 0x7f, 0x40, 0x40, 0x40, 0x40, 0x00,
  0x7f, 0x7f, 0x0e, 0x1c, 0x0e, 0x7f, 0x7f, 0x00,
  0x7f, 0x7f, 0x0e, 0x1c, 0x38, 0x7f, 0x7f, 0x00,
  0x3e, 0x7f, 0x41, 0x41, 0x41, 0x7f, 0x3e, 0x00,
  0x7f, 0x7f, 0x11, 0x11, 0x11, 0x1f, 0x0e, 0x00,
  0x3e, 0x7f, 0x41, 0x51, 0x71, 0x3f, 0x5e, 0x00,
  0x7f, 0x7f, 0x11, 0x31, 0x79, 0x6f, 0x4e, 0x00,
  0x26, 0x6f, 0x49, 0x49, 0x4b, 0x7a, 0x30, 0x00,
  0x00, 0x01, 0x01, 0x7f, 0x7f, 0x01, 0x01, 0x00,
  0x3f, 0x7f, 0x40, 0x40, 0x40, 0x7f, 0x3f, 0x00,
  0x0f, 0x1f, 0x38, 0x70, 0x38, 0x1f, 0x0f, 0x00,
  0x1f, 0x7f, 0x38, 0x1c, 0x38, 0x7f, 0x1f, 0x00,
  0x63, 0x77, 0x3e, 0x1c, 0x3e, 0x77, 0x63, 0x00,
  0x00, 0x03, 0x0f, 0x78, 0x78, 0x0f, 0x03, 0x00,
  0x61, 0x71, 0x79, 0x5d, 0x4f, 0x47, 0x43, 0x00
};

void printbig(int c)
{
  int index = 0;
  int col, data;
  if (c >= '0' && c <= '9') index = 8 + (c - '0') * 8;
  else if (c >= 'A' && c <= 'Z') index = 88 + (c - 'A') * 8;
  do {
    data = font[index++];
    for (col = 0 ; col < 8 ; data <<= 1, col++) {
      char d = data & 0x80 ? 'X' : ' ';
      putchar(d); putchar(d);
    }
    putchar('\n');
  } while (index & 0x7);
}


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

testall.sh:

```
#!/bin/sh

# Regression testing script for compArt
# 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 compArt compiler.  Usually "./compArt.native"
# Try "_build/compArt.native" if ocamlbuild was unable to create a
    symbolic link.
COMPART="./compArt.native"
#COMPART="_build/compArt.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] [.ca files]"
    echo "-k    Keep intermediate files"
    echo "-h    Print this help"
    exit 1
}

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

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

```
    }
}

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

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

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

    echo -n "$basename..."

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

    generatedfiles=""

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

    # Report the status and clean up the generated files

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

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

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

    echo -n "$basename..."

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

    generatedfiles=""

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

    # Report the status and clean up the generated files

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

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

shift `expr $OPTIND - 1`

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

```
}

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

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

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

run.sh:

```
# credit to: Crystal Ren -- Shoo, 2018
#!/bin/bash
set -e
if [ -z "$1" ]
then
echo "Usage: ./run.sh <name_of_file.ca>"
exit 1
fi
f=$1
g="${f//tests}"
h=${g///}
n=${h%.ca*}
cat $f | ./compArt.native > "$n.ll"
llc -relocation-model=pic "$n.ll"
cc -o "$n" "$n.s" compArtHelper.o -L/usr/local/lib -lSDL2 -
    lSDL2_gfx -lSDL2_test -g -O2 -D_THREAD_SAFE -I/usr/local/
    include/SDL2 -I/usr/X11/include -DHAVE_OPENGLES -
    DHAVE_OPENGLES2 -DHAVE_OPENGL -g
rm -r *.ll *.s *.dSYM
"./$n"
```

8.3 Tests

The tests were created by the groups to test the implementations they just created. This is in addition to the given tests, which were edited to fit CompArt: These first tests were designed to fail:

```
/*
./microc.native tests/fail-arrayglobal.mc > fail-arrayglobal.ll
Fatal error: exception Failure("illegal array in global context
    global myarr")
*/

arr[10] myarr;

int main() {
    myarr[0] = 10;
    return 0;
}
```

```
int main() {
    arr myarr; /* error: don't initialize the size */

    return 0;
}
```

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

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

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

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

```
void myvoid()
{
  return;
}

int main()
{
  int i;

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

```
int main()
{
  int i;

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

```
int main()
{
  int i;

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

```
int a;
bool b;

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

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

```
int a;
bool b;

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

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

```
int a;
float b;

```

```
void foo(int c, float d)
{
  int d;
  float e;
  b + a; /* Error: float + int */
}

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

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

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

```
int main()
{
  int i;
  for ( ; true ; ) {} /* OK: Forever */

  for (i = 0 ; i < 10 ; i = i + 1) {
    if (i == 3) return 42;
  }

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

  return 0;
}
```

```
int main()
{
  int i;

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

  return 0;
}
```

```
int main()
{
  int i;

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

  return 0;
}
```

```
int main()
{
  int i;

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

  return 0;
}
```

```
int main()
{
  int i;

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

  return 0;
}
```

```
int foo() {}

int bar() {}

int baz() {}

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

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

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

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

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

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

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

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

```
int foo() {}

void bar() {}

```

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

void baz() {}

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

```
int foo() {}

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

  return 0;
}

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

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

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

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

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

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

void bar()
{
}

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

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

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

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


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

```
int b;
bool c;
int a;
int b; /* Duplicate global variable */

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

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

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

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

```
/* Should be illegal to redefine */
void printb() {}
```

```
/* Should be illegal to redefine */
void printbig() {}
```

```
int main()
{
  return true; /* Should return int */
}
```

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

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

```
int main()
{
  int i;

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

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

}
```

```
int main()
{
  int i;

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

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

}
```

The rest are standard tests that were designed to run fully to check the outputs:

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

int main()
{
  print( add(17, 25) );
```

```
  return 0;
}
```

```
int main()
{
  print(39 + 3);
  return 0;
}
```

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

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

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

```
int main() {
    arr[100] myarr;
    myarr[5] = 10;
    myarr[2] = 12;
    print(myarr[5]);
    return 0;
}
```

```

int main() {
    arr[100] myarr;
    myarr[5] = 10;
    print(myarr[2]=12);
    return 0;
}
```

```

arr[10] getarr() {
    arr[10] myarr;
    myarr[0] = 16;
    myarr[8] = 120;
    print(myarr[8]);
    return myarr;
}


int main() {
```

```
    arr[10] newarr;
    newarr = getarr();
    print(newarr[0]);
    return 0;
}
```

```

arr[10] passarr(arr[10] arr10) {
    return arr10;
}

int main() {
    arr[10] myarr;
    arr[10] newarr;
    int i;
    i = 1;
    myarr[5] = 10;
    myarr[i+1] = 12;
    newarr = passarr(myarr);
    print(newarr[2]);
    return 0;
}
```

```
int main(){
   int i;
   i = 0;
   createWindow(500,500);
   for (;1<2;) {
      color(124, 0, 200);
      drawRect(i, i, 100, 50);
      draw();
      i = i + 1;
   }
   return 0;
}
```

```
int main(){
   int i;
   i = 0;
   createWindow(600,600);
   background(255,255,255);
   for (;1<2;) {
      color(255,255,0);
      drawTriangle(220, 400, 370, 100, 520, 400);
      color(255,0,0);
      drawCircle(260,200,105);
      color(60,179,113);
      drawRect(65,175,250,360);
      draw();
      i = i + 1;
   }
   return 0;
}
```

```
int main(){
   draw(1);
   return 0;
}
```

```
int main() {
   int i;
   createWindow(500,500);
   for (;1<2;) {
      for (i = 0 ; i < 500 ; i = i + 15) {
         color(0,0,255);
         drawLine(i,0,250,250);
         drawLine(500-i,500,250,250);
      }
      draw();
   }
   return 0;
}
```

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

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

```
int main()
{
  float a;
  a = 3.14159267;
  printf(a);
  return 0;
}
```

```
int main()
{
  float a;
  float b;
  float c;
  a = 3.14159267;
  b = -2.71828;
  c = a + b;
  printf(c);
  return 0;
}
```

```
void testfloat(float a, float b)
{
  printf(a + b);
  printf(a - b);
  printf(a * b);
  printf(a / b);
```

```
  printb(a == b);
  printb(a == a);
  printb(a != b);
  printb(a != a);
  printb(a > b);
  printb(a >= b);
  printb(a < b);
  printb(a <= b);
}

int main()
{
  float c;
  float d;

  c = 42.0;
  d = 3.14159;

  testfloat(c, d);

  testfloat(d, d);

  return 0;
}
```

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

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

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

int main()
{
  int a;
  a = add(39, 3);
  print(a);
```

```
  return 0;
}
```

```
/* Bug noticed by Pin-Chin Huang */

int fun(int x, int y)
{
  return 0;
}

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

  fun(i = 2, i = i+1);

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

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

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

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

int main()
{
  int d;
  d = add(52, 10);
  print(d);
  return 0;
}
```

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

int main()
{
```

```
  return 0;
}
```

```
void foo() {}

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

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

```
int a;

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

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

```
void foo(int a)
{
  print(a + 3);
}

int main()
{
  foo(40);
  return 0;
}
```

```
void foo(int a)
{
  print(a + 3);
  return;
}

int main()
{
  foo(40);
  return 0;
}
```

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

```

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

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

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

```
int main(){
   int i;
   setup(500);
   for (;1<2;) {
      for (i = 0 ; i < 250 ; i = i + 10) {
         drawCircle(i);
      }
      draw(1);
   }
   return 0;
}
```

```
int a;
int b;

void printa()
{
  print(a);
}

void printbb()
{
  print(b);
}

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

int main()
{
```

```
  a = 42;
  b = 21;
  printa();
  printbb();
  incab();
  printa();
  printbb();
  return 0;
}
```

```
bool i;

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

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

```
int i;
bool b;
int j;

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

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

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

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

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

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

```
int cond(bool b)
{
  int x;
  if (b)
    x = 42;
  else
    x = 17;
  return x;
}

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

```
int cond(bool b)
{
  int x;
  x = 10;
  if (b)
    if (x == 10)
      x = 42;
  else
    x = 17;
  return x;
}

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

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

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

```

int main()
{
  foo(true);
  return 0;
}
```

```
int foo(int a, bool b)
{
  int c;
  bool d;

  c = a;

  return c + 10;
}

int main() {
 print(foo(37, false));
 return 0;
}
```

```
int main() {
   int i;
   int x;
   int y;
   int r;
   int xspeed;
   int yspeed;
   int window_w;
   int window_h;

   x = 100;
   y = 400;
   xspeed = 1;
   yspeed = 1;
   window_w = 600;
   window_h = 600;

   r = 30;
   createWindow(window_w,window_h);

   for (i = 0;i<765;i = i + 1) {
      color(100, 100, 100);
      drawCircle(x, y, r);
      if (x > window_w-r) xspeed = -xspeed;
      if (x < r) xspeed = -xspeed;
      if (y > window_h-r) yspeed = -yspeed;
      if (y < r) yspeed = -yspeed;
      x = x + xspeed;
      y = y + yspeed;
      draw();
      background(i/3,i/3,i/3);
   }
   return 0;
}
```

```
int main() {
   int i;
   int x;
   int y;
   int r;
   int xspeed;
   int yspeed;
   int window_w;
   int window_h;

   x = 100;
   y = 400;
   xspeed = 1;
   yspeed = 1;
   window_w = 600;
   window_h = 600;

   r = 30;
   createWindow(window_w,window_h);

   for (i = 0;i<765;i = i + 1) {
      color(x, y, y-x);
      drawCircle(x, y, r);
      if (x > window_w-r) xspeed = -xspeed;
      if (x < r) xspeed = -xspeed;
      if (y > window_h-r) yspeed = -yspeed;
      if (y < r) yspeed = -yspeed;
      x = x + xspeed;
      y = y + yspeed;
      draw();
      background(255,255,255);
   }
   return 0;
}
```

```
int main() {
   int i;
   int x;
   int y;
   int r;
   int xspeed;
   int yspeed;
   int window_w;
   int window_h;

   x = 100;
   y = 400;
   xspeed = 1;
   yspeed = 1;
   window_w = 600;
   window_h = 600;

   r = 30;
   createWindow(window_w,window_h);

   for (i = 0;i<765;i = i + 1) {
      color(0, 0, 255);
      opacity(i/3);
```

```
      drawCircle(x, y, r);
      if (x > window_w-r) xspeed = -xspeed;
      if (x < r) xspeed = -xspeed;
      if (y > window_h-r) yspeed = -yspeed;
      if (y < r) yspeed = -yspeed;
      x = x + xspeed;
      y = y + yspeed;
      draw();
      opacity(255);
      background(255,255,255);
   }
   return 0;
}
```

```
int main() {
   int i;
   int x;
   int y;
   int r;
   int xspeed;
   int yspeed;
   int window_w;
   int window_h;

   x = 100;
   y = 400;
   xspeed = 1;
   yspeed = 1;
   window_w = 600;
   window_h = 600;

   r = 30;
   createWindow(window_w, window_h);
   for (;1<2;i = i + 1) {
      background(i/3,i/3,i/3);
      color(x, y, y-x);
      if(x>(window_w/2)) fill(); else noFill();
      drawCircle(x, y, r);
      if (x > window_w-r) xspeed = -xspeed;
      if (x < r) xspeed = -xspeed;
      if (y > window_h-r) yspeed = -yspeed;
      if (y < r) yspeed = -yspeed;
      x = x + xspeed;
      y = y + yspeed;
      draw();
   }
   return 0;
}
```

```
int 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;
}
```

```
int main()
{
  printb(true);
  printb(false);
  printb(true && true);
  printb(true && false);
  printb(false && true);
  printb(false && false);
  printb(true || true);
  printb(true || false);
  printb(false || true);
  printb(false || false);
  printb(!false);
  printb(!true);
  print(-10);
  print(--42);
}
```

```
/*
 * Test for linking external C functions to LLVM-generated code
 *
 * printbig is defined as an external function, much like printf
 * The C compiler generates printbig.o
 * The LLVM compiler, llc, translates the .ll to an assembly .s
    file
 * The C compiler assembles the .s file and links the .o file to
    generate
 * an executable
 */

int main()
{
  printbig(72); /* H */
  printbig(69); /* E */
  printbig(76); /* L */
  printbig(76); /* L */
```

```
  printbig(79); /* O */
  printbig(32); /*   */
  printbig(87); /* W */
  printbig(79); /* O */
  printbig(82); /* R */
  printbig(76); /* L */
  printbig(68); /* D */
  return 0;
}
```

```
int main() {
   int i;
   createWindow(600,600);
   for (;1<2;) {
      for (i = 0 ; i < 600 ; i = i + 30) {
         color(i/3,i/3,i/3);
         drawRect(i,i,i+70,i+70);
      }
      draw();
   }
   return 0;
}
```

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

```
int a;

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

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

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

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

int main()
{
  print(foo(7));
  return 0;
}
```