PIXCZAR

ANIMATION LANGUAGE

Final Report

Frank Aloia - fea2113@columbia.edu - Manager
Gary Chen - gc2676@columbia.edu - System Architect
Bryan Li - bl2557@columbia.edu - Language Guru
Matias Lirman - ml3707@columbia.edu - Tester

Contents

1. Introduction

PixCzar is an object-oriented, imperative programming language that provides users with convenient mechanisms for creating 2D animations. Programmers are able to define and manipulate the motion of created shapes (ellipses, rectangles, and triangles) and uploaded images.

Our language differs from existing animation-based languages because of its unique object-oriented paradigm. Conceptually, the workflow of the language can be likened to a flip-book animation. Programmers specify `Pix` objects, which are individually animated through a series of `Placements`, specifying attributes such as position at a moment in time. These objects are then written to the flip book itself, the `Frame` array, where each `Frame` contains any number of `Placement` objects.

The user will call the `render()` function that will take a Frame[] as an input and initiate a window to display the animation. The object hierarchy and strong references to Pix, Placement, and Frame objects allow for convenient data manipulation. For example, if a Pix is modified, all Frames with a Placement that references this Pix will contain the modified version.

Furthermore, PixCzar's feature set makes it a reasonable general-purpose language. We designed the language with both readability and intuitiveness in mind, and those comfortable with Java should feel right at home with PixCzar. Useful general-purpose features include control flow statements, else if conditional support, and local variable declaration.

2. Language Tutorial

2.1. Installing Dependencies

Ubuntu:
To install OpenGL and SOIL (Simple OpenGL Image Library), run the following command:

```
sudo apt-get install cmake libx11-dev xorg-dev
libglu1-mesa-dev freeglut3-dev libglew1.5 libglew1.5-dev
libglu1-mesa libglu1-mesa-dev libgl1-mesa-glx
libgl1-mesa-dev libsoil-dev
```

That should be all that is needed, but for more details see the tutorial here.

MacOS:
To install OPENGL and SOIL packages on MacOS:

```
brew install glew;
brew install glfw;
git clone https://github.com/kbranigan/Simple-OpenGL-Image-
Library.git SOIL;
cd SOIL;
make;
make install;
```

2.2. Compilation Instructions

All instructions assume a Unix-based system. We have compiled successfully on MacOS 10 (”Darwin”) and Ubuntu 14.04 (”Linux”).

To begin, run `./make.sh`. This generates the PixCzar compiler `pixczar.native` and compiles the OpenGL graphics library `opengl/main.o`.

To execute the test suite, run `./testall.sh`. You may follow along the output to see which tests are being run – green text indicates proper behavior (`SUCCESS` for a passing test, `FAILURE` for a failing one). Red text is improper, but will not be seen, of course.

Compiling an individual .pxr file is straightforward: simply run `./compile.sh <file.pxr>`. This does the following automatically:

1. Replaces `#include` lines if they exist, writing to a new file.
2. Compiles .pxr to .ll using `pixczar.native`
3. Compiles .ll to .s using `llc`
4. Compiles .s to .exe using `clang`, linking the OpenGL, SOIL, and `opengl/main.o` libraries in the process.

Run the executable file `./file.exe` to watch the magic happen (when you wish upon a star...). Alternatively, `./compile.sh <file.pxr> run` compiles and runs at the same time. `./compile.sh <file.pxr> clean` deletes all files created by the original command.

2.3. Quick Tour

This section will walk through the main features of PixCzar by illustrating examples. We start with toy (story) programs, building up our PixCzar knowledge, and end this section with a basic animation program.

2.3.1. Hello World

examples/helloworld.pxr

```
Void main() {
    String s = "Hello World";
    Int x = 123;
    print(s);
    print(x);
}
```

As you can see, `print()` works with `Int` and `String` types. The `main()` function is the entry point to all PixCzar programs.

2.4. Arrays

tests/passing_tests/arrayassign.pxr

```
Void main() {
    Int[] arr = new Int[3];
    arr[0] = 200;
    arr[1] = 200;
    arr[2] = 200;
    print(arr[0]);
}
```

tests/passing_tests/arrayinit.pxr

```
Void main() {
    Int[] arr = [200, 200, 200];
    print(arr[0]);
    print(arr[1]);
    print(arr[2]);
}
```

Here we work with Int[] arrays of length 3 – these are in fact the `rgb` color parameters that a Pix shape takes in (this is grey!). Furthermore, we will soon see how animation is built on Frame[] arrays.

2.5. Loops

tests/passing_tests/continue.pxr

```
Void main() {
  Int x = 10;
  while(--x > 0) {
    if(x % 2 == 0) {
      continue;
    }
  print(x);
  }
}
```

tests/passing_tests/break.pxr

```
Void main() {
  Int x = 1;
  while(x > 0) {
    print(x);
    break;
  }
}
```

tests/passing_tests/iterate_array.pxr

```
Void main() {
  Int[] x = [1,2,3,4,5];
  Int i;
  for(i = 0; i < length(x); i++) {
    print(x[i]);
  }
}
```

These three programs demonstrate looping in PixCzar; `break` exits a loop immediately (break.pxr prints x once), whereas `continue` skips to the next iteration of the loop (continue.pxr prints 9 7 5 3 1 with new lines between them). Also, observe the usage of pre-decrement `--x`, which decrements x then returns that value, as well as the built-in `length()` function.

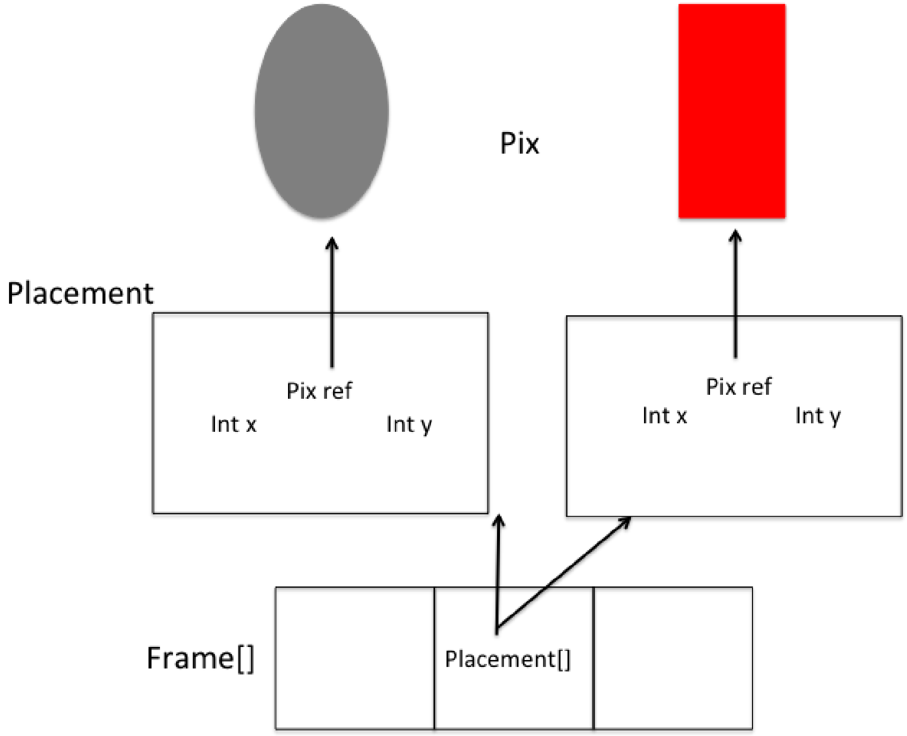
2.6. A Basic Animation

Let's finally put together what we've learned so far with PixCzar animation:

examples/shapes.pxr

```
Void main() {
  Int x;
  Int[] grey = [200,200,200]; // RGB color for grey
  Int[] red = [200,0,0]; // RGB color for red
  Frame[] frames = new Frame[12]; // animation of 12 frames

  // create 100*200 ellipse
  Pix ellipse = new Pix();
  ellipse.makeEllipse(100,200,grey);

  // create 100*200 rectangle
  Pix rect = new Pix();
  rect.makeRectangle(100,200,red);

  // the rectangle moves 50 pixels right every frame
  for(x = 0; x < length(frames); x++) {
    Placement placed1 = new Placement(rect,50*(x+1),100);
    frames[x].addPlacement(placed1);
  }

  // the ellipses move 50 pixels left every 2 frames
  for(x = 0; x < length(frames); x = x + 2) {
    Placement placed = new Placement(ellipse,700-50*(x+1),300);
    frames[x].addPlacement(placed);
    frames[x+1].addPlacement(placed);
  }
  print(render(frames,2,800,600)); // render 800*600 window at 2 fps, then print return code of render()
}
```

What this program does: animates a grey ellipse moving to the right and a red rectangle below it moving to the left twice as fast. Here is the abstract memory diagram of the structures:



Follow along the comments in the code to get the general idea of how animation works. Important takeaways:

- You can think of the Frame[] as the flipbook of PixCzar.
- In each for loop, you are animating a `Pix` by adding a `Placement` corresponding to its position in a `Frame`.
- A `Pix` is immutable and maintains its width, height, and color.

We hope you see from this basic example how PixCzar's object-oriented animation design is both intuitive and powerful. This is just the surface of its functionality– to learn about how you can upload images, utilize keyframes, and be a good PixCzar coder in general, refer to the...

3. Language Reference Manual

This section details all aspects of language syntax and structure in an attempt to give users a resource for creating their own PixCzar animations.

In addition, it discusses the standard library which is an existing library of functions that aim to support programmers in PixCzar development.

3.1. Notation Used in This Manual

The PixCzar Language Reference Manual distinguishes prose, code, and grammar rules by the following font conventions:

- `code` - A monospaced font is used for both code in blocks and for code (e.g. keywords, types) in-line with prose.
- prose - A serif font is used for the content of the LRM itself.
- grammar rules - A sans serif font is used for specifying grammar rules.

3.2. Lexical Conventions

3.2.1. File Extension

PixCzar files must have file extension **.pxr**.

3.2.2. Entry Point

A single function `Void main()` is required. It is the designated entry point of the program.

3.2.3. Identifiers

An identifier is used for declaring variables, and is specified by a sequence of alphabetic characters, underscores, and digits. The first character must be alphabetic, and an identifier cannot be a keyword. There cannot be duplicate identifiers in a scope.

3.2.4. Variable Scope

Each variable only remains in scope in the declared block. If a variable is declared outside a block/function, it is global. A global variable can only be of the following types: Int, Float, String, or Boolean (Section 3.1).

3.2.5. Comments

PixCzar supports both block comments opened by `/*` and closed by `*/`, and single-line comments opened by `//` and closed by a newline character.

3.2.6. Keywords

The following is a list of words reserved by PixCzar. Each of the keywords is further explained in its later relevant section.

category	keywords
basic types	Int, Boolean, Float, Array, String
advanced types	Pix, Placement, Frame, Struct
conditionals	if, else, else if
loops and branching	for, while, return, break, continue
built-in functions	print, length, render, check_access
literals	true, false, null
other	new, Void, include

3.2.7. Global Built-in Functions

- `Void print(Basic_Type output)` - will print any basic type (Section 3.1) to standard out.
- `Void render(Frame[] frames, Int fps, Int width, Int height)` - will take a Frame[] as an input and initiate a window to display the animation at the specified fps rate. `width` and `height` denote the dimensions of the generated window in pixels.
- `Int length(Array arr)` - will return the length of an array of any type.

3.3. Types

All types are capitalized by convention.

3.3.1. Basic types

type name	description
Int	integer number
Float	floating-point number
Boolean	boolean value
String	ordered sequence of ASCII characters

Int.

An integer number in base-10, consists of one or more digits between 0-9.

Float.

A floating point number, consists of digits, a '.', and the decimal part in digits.

Boolean.
A type that evaluates to either `true` or `false`.

String.
An ordered sequence of ASCII characters. Must be enclosed by either single `'` or double `"` quotation marks.

3.3.2. Advanced types
Each instance of an advanced data type has a unique integer hash identifier.

type name	description
Pix	a shape, text box, or image
Placement	a collection of properties of a Pix
Frame	a single frame of animation
Array	a fixed-size collection of same-typed objects

Pix.
Short for a Pixellation, a Pix is any object that can be animated, such as an image, rectangle, triangle, or ellipse.

- Constructor: `Pix()` - new Pix instance
- Built-In Functions (Modifiers for existing Objects)
  - `uploadImage(String path, Int width, Int height)` relative/absolute path to PNG/JPG image, width in pixels, height in pixels. Can accurately rescale images within a factor of 2.
  - `makeEllipse(Int width, Int height, Int[] rgb)` makes ellipse, `width` in pixels, `height` in pixels, `rgb` specifies color and is array `[r, g, b]` with each field between 0-255. Color is undefined if it is out of rgb range.
  - `makeRectangle(Int width, Int height, Int[] rgb)` makes rectangle, `width` in pixels, `height` in pixels, `rgb` specifies color and is array `[r, g, b]` with each field between 0-255
  - `makeTriangle(Int length, Int[] rgb)` makes equilateral triangle, `length` in pixels, `rgb` specifies color and is array `[r, g, b]` with each field between 0-255
  - `clear()` - delete all pixels of Pix

Placement.
A Placement describes the properties of a Pix at a moment in time.

- Constructor: `Placement(Pix ref, Int x, Int y)` - new Placement instance
  - `Pix ref` - reference to a Pix
  - `Int x`, `Int y` - position of Pix on a Frame in pixels

Frame.
A single moment in animation, similar in concept to a page in a flip-book. Holds an array of Placements that defines object positioning on a screen.

- Constructor: `Frame()` - new Frame instance
- Internal Attributes: `Placement[] placed` (List of all Placements existing in the frame)
- Built-In Functions:
  - `addPlacement(Placement place)` - add Placement to `placed` Array of Frames. If placements overlap on the screen, the Placement that was added first will appear ”on top.”
  - `clearPlacements()` - remove all Placements from the Frame

Array.
A fixed-sized collection of same-typed objects. Arrays can contain any type, except internal Arrays.

3.4. Expressions

3.4.1. Precedence and Associativity Rules

Tokens (From High to Low Priority)	Associativity
() [] .	L-R
! ++ - -	R-L
* / %	L-R
+ -	L-R
> >= < <=	L-R
== !=	L-R
&&	L-R
||	L-R
=	R-L
,	L-R

3.4.2. Primary Expressions

Basic building block expressions:

- identifiers
- constants - integer number, floating point number, boolean, string, or null
- advanced types - Pix, Placement, Frame, Array, Struct
- `id( expr* )` - function call with an optional comma-separated list of arguments
- `(expression)` - parenthesized expression

3.5. Unary Operators

- `-(expression)` - evaluates a float or int to the negative value.
- `!(expression)` - logical negation for booleans or conditionals.
- `(expression)++; ++(expression)` - adds 1 to an int; if placed after the int, the value of the expression is the operand's original value; if placed after the int, the value of the expression is the operand's incremented value
- `(expression)--; --(expression)` - subtracts 1 from an int; if placed after the int, the value of the expression is the operand's original value; if placed after the int, the value of the expression is the operand's decremented value
- `~expression` - casts an integer/float to integer. This is especially useful for logical combinations of arithmetic and array access.

3.5.1. Arithmetic Operators

For any (int) <operator>(float), the int will be cast to a float and the operator will return a float. In all other situations, operands must be of the same type.

Operation	Legal Types	Description
`expr1 + expr2`	Int/Float	returns sum
`expr1 - expr2`	Int/Float	returns subtracted value
`expr1 * expr2`	Int/Float	returns multiplied value
`expr1 / expr2`	Int/Float	returns divided value
`expr1 % expr2`	Int	returns modulo of ints

3.6. Relational/Equality Operators

All return booleans. No type casting.

Operation	Legal Types	Description
`expr1 > expr2`	Int/Float	greater than
`expr1 < expr2`	Int/Float	less than
`expr1 >= expr2`	Int/Float	greater than or equal to
`expr1 <= expr2`	Int/Float	less than or equal to
`expr1 == expr2`	Any	equality
`expr1 != expr2`	Any	inequality

Equality comparisons for Pix, Placement, Frame, and Array will compare the memory addresses of the objects.

3.6.1. Assignment Operators

`(id = expr)` will set the variable represented by the identifier to the expression. The variable and the expression must be of the same type. The assignment expression will return the value of the expression.

3.6.2. Array Operators

- `arr[ (idx) ]` - accessing the element of array `arr` referenced by the integer `idx`. Note, if `idx` is an integer literal (and not the return value of a function), the compiler will check for an array out of bounds exception during compilation. Otherwise, the program will check during runtime; it will print a standard out notification and exit the program. It is automatically checked through the built-in function check_access.

3.7. Statements

3.7.1. Expression Statement

Simplest form of statement:

```
( expression );
```

3.7.2. Return Statement

Denotes a value to be returned from a function:

```
return ( optional_expression );
```

If no expression is provided, the return value is Void.

3.7.3. Compound Statement

Allows several statements to be used where one is expected:
`{ statement_list; };`

3.7.4. If Statements

Evaluates condition(s) and executes the statement for the satisfied case.

- `if`: executes a statement if the conditional expression evaluates to a positive (or non `false`, or non `null`) value.
  `if (`*`condition`*`) {`*`statement;`*`}`
- `else if`: any number of `else if` statements are associated with an `if`, and each specifies an alternative condition and statement.
  `if (condition) {statement;} else if (`*`condition`*`) {`*`statement;`*`}`
- `else`: executes a statement if no previous conditional is satisfied; will be connected to the last encountered elseless if
  `if (condition) {statement;} else {`*`statement;`*`}`

3.7.5. Loop Statements

PixCzar allows for both `for` and `while` loops, as well as branching statements `break` and `continue` for further loop control.

- `while`: continually executes code block while conditional expression is a positive (or non `false`, or non `null`) value.
  `while (`*`condition`*`) {`*`statement;`*`}`
- `for`: execute code block a specified number of times. The 3 expressions it takes are *initialization*, which initializes the loop, *termination*, which specifies an exit condition, and *increment*, which is invoked after each iteration of the loop.
  `for(`*`initialization`*`, `*`termination`*`, `*`increment`*`) {`*`statement;`*`}`

3.7.6. Branching Statements

- `break`: terminate the current (inner-most) loop. Note: loops consist of `for` and `while` iterations, and do not include `if` or `else` iterations.
- `continue`: skip the current iteration of a loop. Afterwards, the loop condition will be re-evaluated for the next iteration.

3.7.7. Declaration Statements

Default Type Values.

Default values for each variable type. These represent the value of a variable after its declared with no expression:

Type	Default Value
Int	0
Float	0.0
String	””
Boolean	false
Pix	new Pix()
Placement	new Placement(new Pix(), 0, 0)
Frame	new Frame()
Array	new <Type >[0]

Basic Type Declaration.

To declare a variable id with a basic type (expression must be of type `typ`):

```
typ id = ( expression );
```

Pix/Placement/Frame Declaration.

All variables of types Pix, Placements, or Frames are strong references to objects. To set a variable id to an existing object referenced by variable obj:

```
type id = obj;
```

To declare new objects, use the `new` keyword with the object constructor:

```
type id = new type( expr1, expr2, ...);
```

Array Declaration.

To declare a new Array of type `typ` and size `size`:

```
typ id = new typ[size];
```

Values can also be initialized during declaration:

```
Int arr = [0, 1, 2, 3];
```

If not initialized, each element will be the default type value.

Multiple Declarations in a Line.

Multiple variables of the same type can be declared (and optionally initialized) in a single line. They are separated by commas:

```
Int x, y = 0, z;
```

3.7.8. Advanced-Type Function Calls

Pix, Placements, and Frames have built-in function calls that modify existing objects:

```
obj.func(expr*);
```

This will call a function func with the specified arguments on the object obj. Object function calls do not create new objects. There is not return value.

3.8. Additional Control Flow

3.8.1. Functions

Functions have any number of input arguments and a single return type. If the return type is Void, the function does not require a return statement. If the return type is a basic type and no return statement is provided, the function will return the default value of the return type (Section 3.7.7). Function names follow the same rules as regular identifiers. There are no nested functions or high-order functions. They are declared as follows:

```
Type Function_Name(expr*){ statement_list; }
```

3.8.2. Linking Files

The contents of other .pxr files can be linked to the working file. There can only be one `Void main()` function among linked files. Files are linked using the `include` keyword: `#include "<pathname>"`.

Pathname is a string denoting the relative path in the directory. The `include` keyword can be used in a statement at any point in the file and the contents of the linked file will replace the `include` statement. The linking logic is implemented in a bash script labeled ”include.sh.” Here are the steps to generate a file with the included logic:

1. `source ./include.sh` from the top level directory.
2. `generate_includes` <filename>will return a new file <filename>_included.pxr

If you are compiling manually, remember to compile <filename>_included.pxr and not the original file. If running `compile.sh`, there is no need; it will handle it for you.

3.9. Standard Library

The standard library is currently a list of functions in file ”stdlib.pxr” (see above ”Linking Files” section on include syntax).

- `Void fillFrames(Frame[] frames, Placement placement, Int start, Int end` - add a Placement to every Frame in an Array from index `start` until `end`
- `Void keyFrame(Frames[] frames, Int start, Pix obj, Int[] from, Int[] to, Int duration)` - adds Placements to `frames` starting at index `start`, making `obj` move from point `from` to point `to` over a specified duration
- `Int[] projectile(Frame[] frames, Int start, Pix obj, Int xSpeed, Int width, Int height, Int endHeight, Int gravity` - function specifically created to help an object curve to a particular location. Adds Placements to `frames` starting at index `start`, making object `obj` move at `xSpeed` pixels/Frame and vertical acceleration `gravity` until reaching the `endHeight`. Often used for creating ”quarter circles” or arcs. Vertical acceleration can be positive or negative.

3.9.1. Sample Code

Bouncing Ball.

Simulates the animation of a ball bouncing up and down:

```
Void main() {
    Frame[] framesReel = new Frame[6];
    Int i;     for(i = 0; i < 6; i++) {
        framesReel[i] = new Frame();
        /* Creating new Frames */
    }

    Pix ball = new Pix();
    ball.makeEllipse(2, 2, [255,255,255]);

    Placement p1 = new Placement(ball, 5, 5);
    Placement p2 = new Placement(ball, 5, 0);
    for(i = 0; i < length(framesReel) ; i++) {
        if (i % 2 == 0) {
            framesReel[i].addPlacement(p1);
        } else{
            framesReel[i].addPlacement(p2);
        }
    }
    render(framesReel, 30, 800, 600);

    }
```

3.9.2. Fill Frames

Fill Frames Library Function:

```
Void fillFrames(Frame[] frames, Placement placement, Int
start, Int end) {
    Int i;
    for(i = start; i < end; i++) {
        if(i < length(frames)) {
            frames.addPlacement(placement);
        }
    }
}
```

3.9.3. Key Frames

Key Frame Library Function:

```
Void keyFrame(Frames[] frames, Int start, Pix obj, Int[]
from, Int[] to, Int duration) {
    Float xTimeStep = (to[0]-from[0])/duration;
    Float yTimeSTep = (to[1]-from[1])/duration;

    Int i;
    for (i=0; i < length(frames); i++) {
        Placement plcmt = new Placement(
            obj, from[0] + i*xTimeStep, from[y] + i*yTimestep);
        if(i + start < length(frames)) {
            frames[i+start].addPlacement(plcmt);
        }
    }
}
```

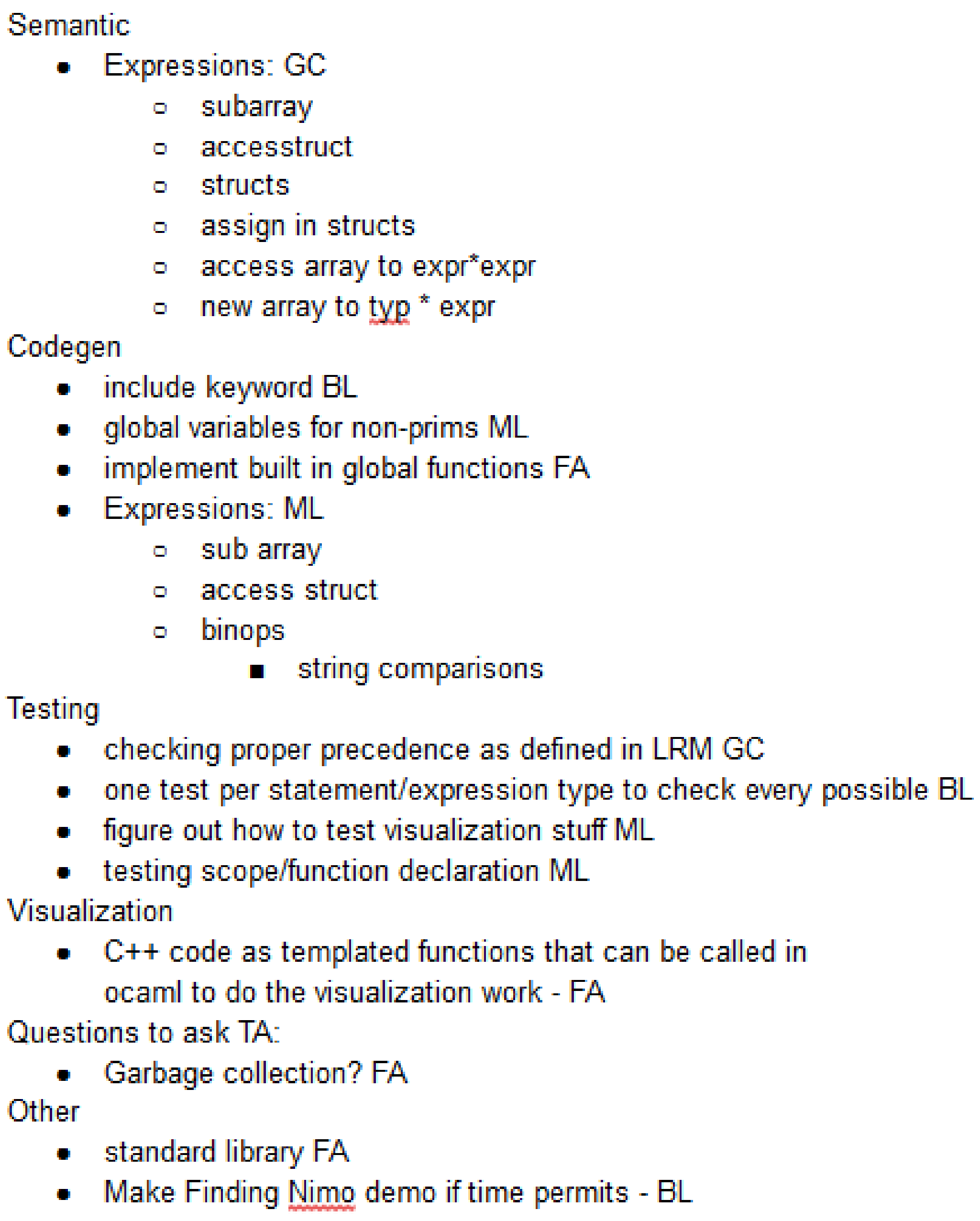
4. Project Planning

4.1. Process

We had a two-tiered working process. First, we split off into pairs, Gary + Bryan, Frank + Mat (on account of living together), and each met throughout the week. As a whole group, we tried to meet once a week to sync up on progress. As the manager, Frank would compile a list of current tasks and check in with each member. Mediating between small and large group meetings was our Slack channel, where communication was constant and we asked quick questions about design implementation, and code review.

We worked alongside the project deliverables and found that the four of them were quite adequate in keeping us at a good developmental pace. For the planning stage of the project, the four of us met multiple times a week to decide on a cohesive product vision. As time moved along, and we settled into our roles, we were able to work more independently.

In terms of specification and development, we used a collaborative document to outline all the current tasks, initialing items to indicate that a person(s) was working on it. Everyone was able to see the progress, as well as what tasks to take on next. At each checkpoint, we removed completed tasks. Here is a copy of a WIP version of this document (this is before everything was completed):



Semantic

- Expressions: GC
  - subarray
  - accesstruct
  - structs
  - assign in structs
  - access array to expr*expr
  - new array to typ * expr

Codegen

- include keyword BL
- global variables for non-prims ML
- implement built in global functions FA
- Expressions: ML
  - sub array
  - access struct
  - binops
    - string comparisons

Testing

- checking proper precedence as defined in LRM GC
- one test per statement/expression type to check every possible BL
- figure out how to test visualization stuff ML
- testing scope/function declaration ML

Visualization

- C++ code as templated functions that can be called in ocaml to do the visualization work - FA

Questions to ask TA:

- Garbage collection? FA

Other

- standard library FA
- Make Finding Nimo demo if time permits - BL

4.2. Style Guide

Here are some basic guidelines for our OCaml code:

- Line break after the keyword 'in'

- Two spaces to denote a new level of name bindings
- Include parenthesis around tuples
- Include parenthesis around every pattern-matching statement
- Unless all patterns can fit on one line, line break between each pattern. Include a four space indent to begin each pattern name. Ensure all vertical bars are aligned.
- Prioritize readability over a strict character limit per line
- Include parenthesis around if statements
- Always use recursive behavior, as opposed to C-like looping mechanisms
- Minimize the number of global/mutable variables
- Follow the general principle of minimizing duplicated logic

4.3. Project Timeline

We used the four project deliverables as a baseline. We also set additional checkpoints. Overall, we were more productive after the Hello World deliverable. By the end of classes, we ended up ahead of schedule for the basic implementation. The following is a rough timeline:

Date	Default Value
February 2	Proposal deliverable due, roles assigned
February 16	Developed scanner and parser
February 21	Deliverable #1, developed AST and AST tests
February 26	Deliverable #2, wrote LRM
March 7	Developed SAST, start semantic checker and codegen
March 27	Deliverable #3, full stack working for hello world
April 8	Developed assignment and variable declaration
April 15	Developed arrays, extend testsuite
April 17	Figured out OpenGL linking
April 20	Deliverable #4, implement conditionals
April 27	Implemented `Pix` shapes
May 2	Wrote compile, make, include scripts
May 5	Implemented `uploadImage` using SOIL
May 8	Final report written, finishing up touches
May 9	Final Presentation
May 16	The great PixCzar graduation

4.4. Roles and Responsibilities

Frank Aloia - Manager
In charge of:

- Assigning tasks, main communication point, all organization and progress tracking
- Graphics pipeline, from linking libraries to code generation
- Arrays and Objects
- Standard library
- LLVM code generation
- Demo

Helped with:

- AST, SAST, semantic checker
- Language design decisions
- Documentation for LRM and Final Report

Gary Chen - System Architect In charge of:

- Scanner and parser
- Finalizing test suite
- System architecture and design
- Final presentation

Helped with:

- Final report content
- SAST, semantic checker

Bryan Li - Language Guru
In charge of:

- Language Reference Manual, Final Report
- Creating include, compile scripts
- Image editing for examples

Helped with:

- SAST, code generation
- Assignment, system architecture

Matias Lirman - Tester In charge of:

- Test suite and test flow
- Semantic checker
- SAST and AST

Helped with:

- Arrays and objects
- Graphics pipeline

4.5. Software Development Environment

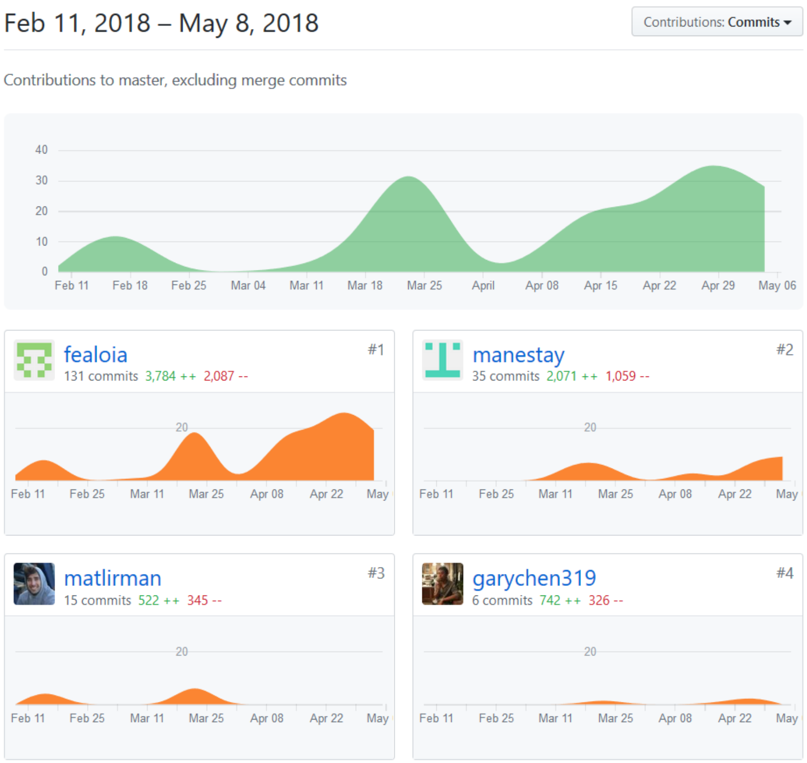
Frank, Gary, and Mat used MacOS, while Bryan used Ubuntu. As both are Unix-like environments, there were no major differences, aside from different compilation flags.

Ubuntu environment: Ubuntu 14.04, OCaml 3.4, clang 3.4, OpenGL 1.5, Geany text editor
MacOS environment: OCaml 4.06.1, Apple LLVM 9.0.0, OPENGL 2.1
Both environments: Soil 10.36, Github, Slack

4.6. Project Log

Note: As stated above, we did a lot of work in two groups. Within groups we mainly worked on one PC (Bryan's and Frank's especially). Everyone in the group contributed to development. Exact number of commits and lines of code are not necessarily reflective of participation or effort, as tasks differ in complexity and verbosity.

Figure 1: Contributors



Feb 11, 2018 – May 8, 2018

Contributions: Commits ▾

Contributions to master, excluding merge commits

fealoia #1
131 commits 3,784 ++ 2,087 --

manestay #2
35 commits 2,071 ++ 1,059 --

matlirman #3
15 commits 522 ++ 345 --

garychen319 #4
6 commits 742 ++ 326 --

Figure 2: Code Frequency

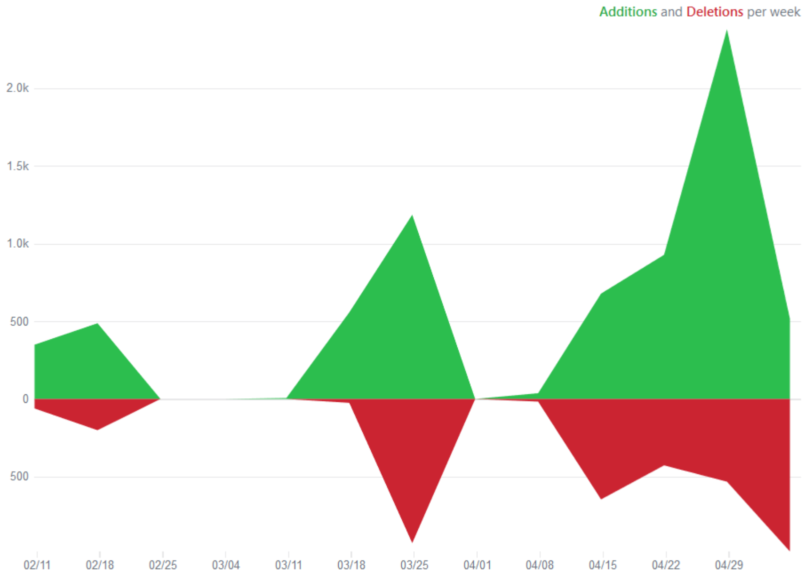
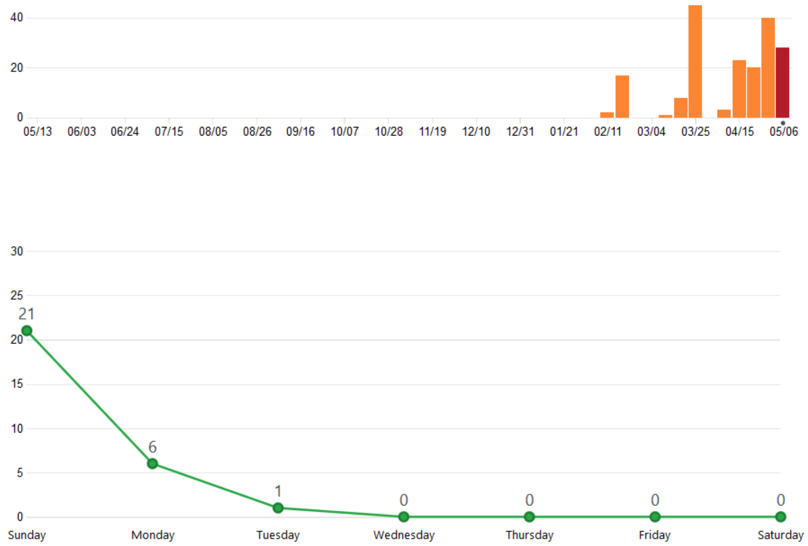


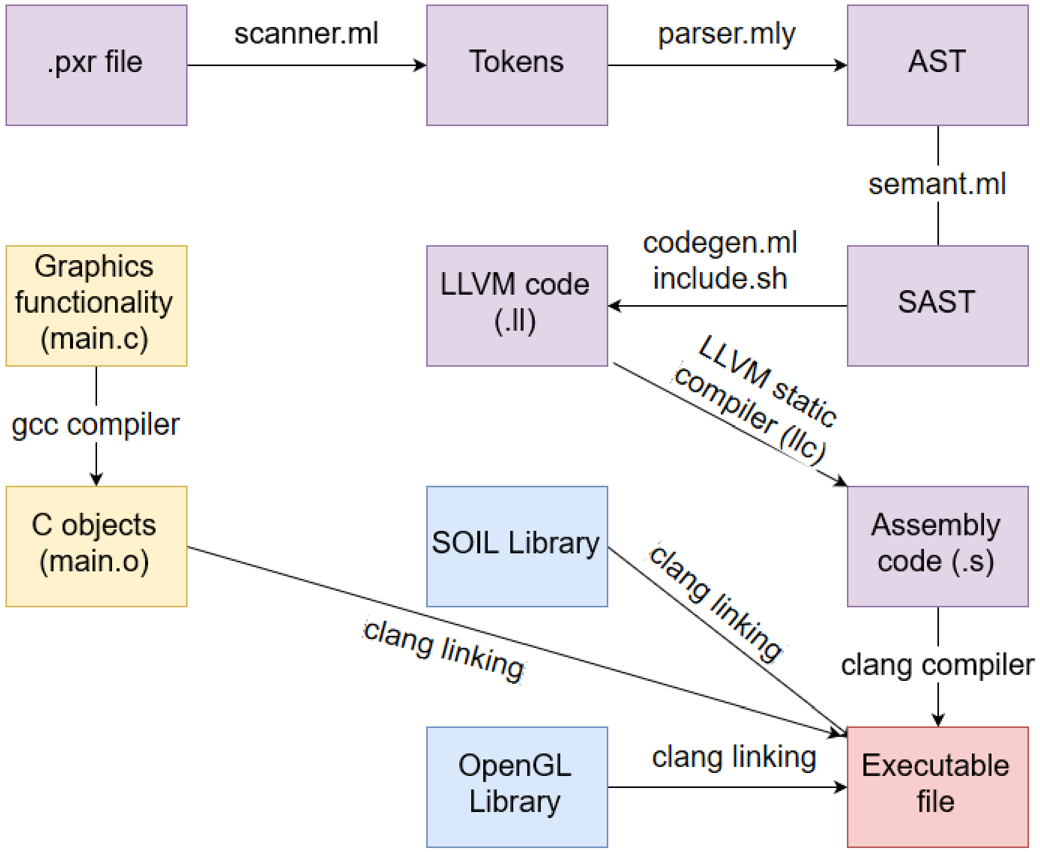
Figure 3: Commit Statistics



For readability purposes, we chose not to include the log of each individual git commit. If interested, please view the following link:
https://github.com/fealoia/pixczar/commits/master

5. Architectural Design

5.1. Design Diagram



5.2. Interfaces between Components

5.2.1. Scanner

The scanner (scanner.ml) reads the source codes from the .pxr file and converts the sequence of characters into tokens identifiable by our language, such as keywords, identifiers and literals. It strips the code of unnecessary whitespace and comments and leaves only the tokens necessary for the parser.

5.2.2. Parser

The parser (parser.mly) receives tokens from the scanner and checks that the tokens given are syntactically accurate. An abstract syntax tree (AST) is created using the tokens as nodes.

5.2.3. Semantic Checker

The semantic checker (semant.ml) takes an AST and generates a syntactically checked abstract syntax tree, ensuring that the program follows the conventions of the PixCzar language.

5.2.4. Code Generator

The code generator (codegen.ml) traverses through the SAST and uses the tokens to generate LLVM byte code.

5.2.5. Graphics Functionality

Our graphics are implemented using C. In the code generator, each Frame is implemented as a struct, which only contains a pointer to a placement node. A placement node contains a pointer to a Placement and the pointer to the next placement node. Every time a Placement is added to a Frame, the encapsulating node becomes the head of the linked list. The render() function in C will take the frame array as an input. The only logic implemented in C is iterating through the placement node in each Frame and calling the required openGL function. In order to display the graphics that our executable file requires, we link the SOIL and OPENGL Libraries using clang. The executable is compiled into a .exe from the assembly code and uses these libraries to display the graphics on a window.

5.3. Who Implemented What?

Feature	Main Developer(s)	Assistant(s)
scanner.mll	Gary	Frank, Mat
parser.mly	Gary, Frank	Mat
ast.ml	Mat	Frank, Bryan
sast.ml	Mat, Bryan	Frank
semant.ml	Gary, Bryan, Frank	Mat
codegen.ml	Frank	Bryan, Gary, Mat
main.c	Frank	
compile.sh	Bryan	
testall.sh	Mat, Gary	Bryan, Frank

6. Test Plan

6.1. Overview

We used unit testing to ensure our scanner was properly defined and our parser correctly implemented the grammar and parsed each token correctly.

The code generator was tested by compiling the program and examining the LLVM IR code output and also verifying whether the expected output was produced. Each part of the architecture was tested as it was being developed.

The integration tests were split into two categories of passing and failing tests. The passing tests were straightforward to implement, for each function of our language, we wrote a test case and checked its output against the expected output file. For each test, we would also include all valid edge cases such as 0, negatives, different types, different scopes, etc.

For failure cases, we focused on edge cases that are not allowed in the language. Common examples include negative numbers, invalid types, incorrect number of arguments, out of scope variables, etc. Each function generates multiple test cases for each invalid edge case.

6.2. Scanner & Parser Unit Testing

Our language supports usage of custom flags at the end of compile commands to test specific parts of the architecture. The ”-a” flag prints the AST, the ”-s” flag prints the SAST, the ”-l” flag prints the generated LLVM IR, and the ”-c” performs the default compile action. A PixCzar program can be compiled as ”./pixczar.native [-a/-s/-l/-c] [file.pxr]” with the desired flag for unit testing purposes.

6.3. Automation

The test suite is comprised of 145 test files, 67 pass cases and 78 fail cases. The positive test cases are designed to check the validity of a variable declaration, operator, or function, and will return positive if the produced output matches the intended output. The negative cases are designed to break the code, and will return positive if the appropriate error message is returned. Each test is comprised of a .pxr file and a .out file; the .pxr file contains the code content of the text and the .out file is the output that the test should match in order to be considered a passing test. A diff command can easily be performed between the output of each .pxr program and its corresponding .out file to confirm the test results.

6.4. Test Scripts

testall.sh (simply run using ./testall.sh) is a script that automatically runs every .pxr file in the ”tests” folder and checks for differences between its output and the corresponding .out file. If the test succeeds the script will print to the console the name of the test, and SUCCESS/FAILURE (depending on whether it's a intended passing or failing test) in green letters. If the tests fails, then the script will print to the console the name of the test, the output of the .pxr test, followed by the expected output, and FAILURE/SUCCESS OR NOT FOUND in red letters.

6.5. Example Test Cases

6.5.1. Sample Passing Test Case

operator_precedence.pxr

```
Void main(){
    Int x = (1+(2+3)*4);
    print(x);
}
```

operator_precedence.out:

```
21
```

This is a simple test program that checks the program's arithmetic operators and precedence. Note, integer declaration and the print function have already been checked in previous tests. In this arithmetic expression, the operator inside the brackets (2+3) is supposed to be evaluated first. This yields 1+5*4, and the 5*4 should be evaluated next, due to the precedence of the * operator. Finally 1+20 should yield the correct result 21.

If the program had ignored parentheses precedence, then the value of x would have yielded 15. If it had ignored precedence altogether, it would have yielded a result of 24. Further tests that isolate parentheses or multiplication/division operators can be implemented to verify this result.

6.5.2. Sample Failing Test Case

outofscope.pxr

```
Void Hello(){
    print(x);
}
```

```
Void main() {
    Int x = 3;
    Hello();
}
```

outofscope.out

```
Fatal error: exception Failure("undeclared identifier x")
```

This test case is designed to throw an error due to an undeclared variable. main() declares `Int x = 3` and proceeds to call `Hello()` without passing any arguments. `Hello()` then attempts to `print(x)`, but x was never declared in the current function. The .out file contains the intended error ”undeclared identifier x,” since the variable x should never have been declared in the scope of the Hello() function.

6.6. Generated Code Examples

Below are generated code examples to use as reference for testing

6.6.1. Placing Rectangle Bottom Left Corner

```
Void main() {
  Int x;
  Int[] grey = [200,200,200]; // RGB color for grey
  Frame[] frames = new Frame[1];

  // create 100*200 rectangle
  Pix rect = new Pix();
  rect.makeRectangle(100,200,grey);

  // Place rect in bottom left corner of screen
  Placement placed1 = new Placement(rect,0,0);
  frames[x].addPlacement(placed1);

  render(frames,2,800,600); // render on 800*600 window at 2 frames
     per second
}
```

```
; ModuleID = 'PixCzar'
```

```
%frame = type { %placement_node* }
%placement_node = type { %placement_node*, %placement* }
%placement = type { %pix*, i32, i32 }
%pix = type { i32, i8*, i32, i32, i32* }

@tmp = private unnamed_addr constant [50 x i8] c"Throwing Runtime
   Error: Out of Bound Array Access\00"
@fmt = private unnamed_addr constant [4 x i8] c"%s\0A\00"

declare void @exit(...)

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

declare i32 @render(i32, %frame**, i32, i32, i32, ...)

define void @main() {
entry:
  %x = alloca i32
  store i32 0, i32* %x
  %malloccall = tail call i8* @malloc(i32 mul (i32 ptrtoint (i32*
     getelementptr (i32, i32* null, i32 1) to i32), i32 4))
  %array_gen = bitcast i8* %malloccall to i32*
  %size = getelementptr i32, i32* %array_gen, i32 0
  store i32 3, i32* %size
  %array_assign = getelementptr i32, i32* %array_gen, i32 1
  store i32 200, i32* %array_assign
  %array_assign1 = getelementptr i32, i32* %array_gen, i32 2
  store i32 200, i32* %array_assign1
  %array_assign2 = getelementptr i32, i32* %array_gen, i32 3
  store i32 200, i32* %array_assign2
  %grey = alloca i32*
  store i32* %array_gen, i32** %grey
  %malloccall.3 = tail call i8* @malloc(i32 mul (i32 ptrtoint (i1**
     getelementptr (i1*, i1** null, i32 1) to i32), i32 2))
  %array_gen4 = bitcast i8* %malloccall.3 to %frame**
  %size5 = getelementptr %frame*, %frame** %array_gen4, i32 0
  %cast = bitcast %frame** %size5 to i32*
  store i32 1, i32* %cast
  %array_assign6 = getelementptr %frame*, %frame** %array_gen4, i32
     1
```

```
%malloccall.7 = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64
    ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i64),
    i64 2) to i32))
%malloc = bitcast i8* %malloccall.7 to %placement_node*
%struct_build = getelementptr inbounds %placement_node,
    %placement_node* %malloc, i32 0, i32 0
store %placement_node* null, %placement_node** %struct_build
%struct_build8 = getelementptr inbounds %placement_node,
    %placement_node* %malloc, i32 0, i32 1
store %placement* null, %placement** %struct_build8
%malloccall.9 = tail call i8* @malloc(i32 ptrtoint (i1**
    getelementptr (i1*, i1** null, i32 1) to i32))
%malloc10 = bitcast i8* %malloccall.9 to %frame*
%struct_build11 = getelementptr inbounds %frame, %frame*
    %malloc10, i32 0, i32 0
store %placement_node* %malloc, %placement_node** %struct_build11
store %frame* %malloc10, %frame** %array_assign6
%frames = alloca %frame**
store %frame** %array_gen4, %frame*** %frames
%malloccall.12 = tail call i8* @malloc(i32 ptrtoint (%pix*
    getelementptr (%pix, %pix* null, i32 1) to i32))
%malloc13 = bitcast i8* %malloccall.12 to %pix*
%struct_build14 = getelementptr inbounds %pix, %pix* %malloc13,
    i32 0, i32 0
store i32 0, i32* %struct_build14
%struct_build15 = getelementptr inbounds %pix, %pix* %malloc13,
    i32 0, i32 1
store i8* null, i8** %struct_build15
%struct_build16 = getelementptr inbounds %pix, %pix* %malloc13,
    i32 0, i32 2
store i32 0, i32* %struct_build16
%struct_build17 = getelementptr inbounds %pix, %pix* %malloc13,
    i32 0, i32 3
store i32 0, i32* %struct_build17
%struct_build18 = getelementptr inbounds %pix, %pix* %malloc13,
    i32 0, i32 4
store i32* null, i32** %struct_build18
%rect = alloca %pix*
store %pix* %malloc13, %pix** %rect
%rect19 = load %pix*, %pix** %rect
```

```
%grey20 = load i32*, i32** %grey
%struct_build21 = getelementptr inbounds %pix, %pix* %rect19, i32
   0, i32 0
store i32 1, i32* %struct_build21
%struct_build22 = getelementptr inbounds %pix, %pix* %rect19, i32
   0, i32 1
store i8* null, i8** %struct_build22
%struct_build23 = getelementptr inbounds %pix, %pix* %rect19, i32
   0, i32 2
store i32 100, i32* %struct_build23
%struct_build24 = getelementptr inbounds %pix, %pix* %rect19, i32
   0, i32 3
store i32 200, i32* %struct_build24
%struct_build25 = getelementptr inbounds %pix, %pix* %rect19, i32
   0, i32 4
store i32* %grey20, i32** %struct_build25
%rect26 = load %pix*, %pix** %rect
%malloccall.27 = tail call i8* @malloc(i32 ptrtoint (%placement*
   getelementptr (%placement, %placement* null, i32 1) to i32))
%malloc28 = bitcast i8* %malloccall.27 to %placement*
%struct_build29 = getelementptr inbounds %placement, %placement*
   %malloc28, i32 0, i32 0
store %pix* %rect26, %pix** %struct_build29
%struct_build30 = getelementptr inbounds %placement, %placement*
   %malloc28, i32 0, i32 1
store i32 0, i32* %struct_build30
%struct_build31 = getelementptr inbounds %placement, %placement*
   %malloc28, i32 0, i32 2
store i32 0, i32* %struct_build31
%placed1 = alloca %placement*
store %placement* %malloc28, %placement** %placed1
%frames32 = load %frame**, %frame*** %frames
%x33 = load i32, i32* %x
%size34 = getelementptr %frame*, %frame** %frames32, i32 0
%cast35 = bitcast %frame** %size34 to i32*
%size36 = load i32, i32* %cast35
call void @check_access(i32 %x33, i32 %size36)
%add = add i32 %x33, 1
%arr_access = getelementptr %frame*, %frame** %frames32, i32 %add
%arr_access_val = load %frame*, %frame** %arr_access
```

```
  %placed137 = load %placement*, %placement** %placed1
  %add_plcmt = getelementptr inbounds %frame, %frame*
     %arr_access_val, i32 0, i32 0
  %node = load %placement_node*, %placement_node** %add_plcmt
  %malloccall.38 = tail call i8* @malloc(i32 trunc (i64 mul nuw
     (i64 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to
     i64), i64 2) to i32))
  %malloc39 = bitcast i8* %malloccall.38 to %placement_node*
  %struct_build40 = getelementptr inbounds %placement_node,
     %placement_node* %malloc39, i32 0, i32 0
  store %placement_node* %node, %placement_node** %struct_build40
  %struct_build41 = getelementptr inbounds %placement_node,
     %placement_node* %malloc39, i32 0, i32 1
  store %placement* %placed137, %placement** %struct_build41
  store %placement_node* %malloc39, %placement_node** %add_plcmt
  %frames42 = load %frame**, %frame*** %frames
  %render = call i32 (i32, %frame**, i32, i32, i32, ...)
     @render(i32 1, %frame** %frames42, i32 2, i32 800, i32 600)
  ret void
}

define void @check_access(i32, i32) {
entry:
  %param_alloc = alloca i32
  store i32 %0, i32* %param_alloc
  %param_alloc1 = alloca i32
  store i32 %1, i32* %param_alloc1
  %tmp = icmp sgt i32 %1, %0
  br i1 %tmp, label %checked, label %exit

checked:                                     ; preds = %entry
  ret void

exit:                                        ; preds = %entry
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
     ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i8* getelementptr
     inbounds ([50 x i8], [50 x i8]* @tmp, i32 0, i32 0))
  call void (...) @exit()
  ret void
}
```

```
declare noalias i8* @malloc(i32)
```

6.6.2. Position of Projectile

```
//Calculate trajectory of a canon ball
//Scale 1 pixels = 1 meter

Int distanceTravelled(Int t) {
    // d = 1/2gt^2
    // gravity = 9.8 m/s^2
    return 5 * t * t;
}

Void main() {
    Int[] x = ballTrajectory(5, 50, 150, 250);
    print(x[0]);
}

/* Calculating position on window [x,y] after 5 frames of
traveling at this trajectory */

Int[] ballTrajectory(Int frames, Int xSpeed, Int width, Int
    height) {
    Int secondsPerFrame = 1;

    Int d = distanceTravelled(secondsPerFrame);

    Int i;
    for(i=0; i < frames; i++) {
        width = width + i*xSpeed*secondsPerFrame;
        height = height - d;
        d = distanceTravelled(secondsPerFrame*i);
    }
    Int[] pos = [width, height];
    return pos;
}
```

```
; ModuleID = 'PixCzar'
```

```
%frame = type { %placement_node* }
%placement_node = type { %placement_node*, %placement* }
%placement = type { %pix*, i32, i32 }
%pix = type { i32, i8*, i32, i32, i32* }

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@tmp = private unnamed_addr constant [50 x i8] c"Throwing Runtime
   Error: Out of Bound Array Access\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%s\0A\00"

declare void @exit(...)

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

declare i32 @render(i32, %frame**, i32, i32, i32, ...)

define void @main() {
entry:
  %ballTrajectory = call i32* @ballTrajectory(i32 5, i32 50, i32
      150, i32 250)
  %x = alloca i32*
  store i32* %ballTrajectory, i32** %x
  %x1 = load i32*, i32** %x
  %size = getelementptr i32, i32* %x1, i32 0
  %size2 = load i32, i32* %size
  call void @check_access(i32 0, i32 %size2)
  %arr_access = getelementptr i32, i32* %x1, i32 1
  %arr_access_val = load i32, i32* %arr_access
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32 %arr_access_val)
  ret void
}

define void @check_access(i32, i32) {
entry:
  %param_alloc = alloca i32
  store i32 %0, i32* %param_alloc
  %param_alloc1 = alloca i32
  store i32 %1, i32* %param_alloc1
  %tmp = icmp sgt i32 %1, %0
  br i1 %tmp, label %checked, label %exit

checked:                                      ; preds = %entry
  ret void

exit:                                         ; preds = %entry
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.1, i32 0, i32 0), i8* getelementptr
      inbounds ([50 x i8], [50 x i8]* @tmp, i32 0, i32 0))
  call void (...) @exit()
  ret void
}

define i32* @ballTrajectory(i32, i32, i32, i32) {
entry:
  %param_alloc = alloca i32
  store i32 %0, i32* %param_alloc
  %param_alloc1 = alloca i32
  store i32 %1, i32* %param_alloc1
  %param_alloc2 = alloca i32
  store i32 %2, i32* %param_alloc2
  %param_alloc3 = alloca i32
  store i32 %3, i32* %param_alloc3
  %secondsPerFrame = alloca i32
  store i32 1, i32* %secondsPerFrame
  %secondsPerFrame4 = load i32, i32* %secondsPerFrame
  %distanceTravelled = call i32 @distanceTravelled(i32
      %secondsPerFrame4)
  %d = alloca i32
  store i32 %distanceTravelled, i32* %d
  %i = alloca i32
  store i32 0, i32* %i
  store i32 0, i32* %i
  br label %while

while:                                        ; preds =
    %while_body, %entry
  %i18 = load i32, i32* %i
  %frames = load i32, i32* %param_alloc
  %tmp19 = icmp slt i32 %i18, %frames
  br i1 %tmp19, label %while_body, label %merge

merge:                                        ; preds = %while
  %malloccall = tail call i8* @malloc(i32 mul (i32 ptrtoint (i32*
      getelementptr (i32, i32* null, i32 1) to i32), i32 3))
  %array_gen = bitcast i8* %malloccall to i32*
  %size = getelementptr i32, i32* %array_gen, i32 0
  store i32 2, i32* %size
  %array_assign = getelementptr i32, i32* %array_gen, i32 1
  %width20 = load i32, i32* %param_alloc2
  store i32 %width20, i32* %array_assign
  %array_assign21 = getelementptr i32, i32* %array_gen, i32 2
  %height22 = load i32, i32* %param_alloc3
  store i32 %height22, i32* %array_assign21
  %pos = alloca i32*
  store i32* %array_gen, i32** %pos
  %pos23 = load i32*, i32** %pos
  ret i32* %pos23

while_body:                                   ; preds = %while
  %width = load i32, i32* %param_alloc2
  %i5 = load i32, i32* %i
  %xSpeed = load i32, i32* %param_alloc1
  %tmp = mul i32 %i5, %xSpeed
  %secondsPerFrame6 = load i32, i32* %secondsPerFrame
  %tmp7 = mul i32 %tmp, %secondsPerFrame6
  %tmp8 = add i32 %width, %tmp7
  store i32 %tmp8, i32* %param_alloc2
  %height = load i32, i32* %param_alloc3
  %d9 = load i32, i32* %d
  %tmp10 = sub i32 %height, %d9
  store i32 %tmp10, i32* %param_alloc3
  %secondsPerFrame11 = load i32, i32* %secondsPerFrame
  %i12 = load i32, i32* %i
  %tmp13 = mul i32 %secondsPerFrame11, %i12
  %distanceTravelled14 = call i32 @distanceTravelled(i32 %tmp13)
  store i32 %distanceTravelled14, i32* %d
  %i15 = load i32, i32* %i
  %i16 = load i32, i32* %i
  %tmp17 = add i32 %i16, 1
  store i32 %tmp17, i32* %i
  br label %while
}

define i32 @distanceTravelled(i32) {
entry:
  %param_alloc = alloca i32
  store i32 %0, i32* %param_alloc
  %t = load i32, i32* %param_alloc
  %tmp = mul i32 5, %t
  %t1 = load i32, i32* %param_alloc
  %tmp2 = mul i32 %tmp, %t1
  ret i32 %tmp2
}
declare noalias i8* @malloc(i32)
```

6.7. Who did What?

Mat was the primary driver for the testsuite, creating the testing scripts and automation. Gary, secondarily, contributed. All team members wrote additional tests during development. The person implementing a feature would write the corresponding tests.

7. Lessons Learned

Frank: It is crucial that group members lay a good foundation for their project from the very start. This includes communication between team members, maintaining a healthy group dynamic, and setting a regular meeting schedule. Even in the early stages, it is important to have a clear vision of the project that every member of the group understands and agrees on. A project that all group members are passionate about will ultimately be more successful. Seemingly insignificant things such as getting to know each group member and setting up group chats also help quite a bit, as it ensures that everyone is friendly and isnt speak up about their different ideas. We learned that although our preliminary brainstorming sessions of the project may have seemed like small-talk, they actually played a huge role in laying the groundwork for our team dynamic.

Bryan: The actual planning of the project is just as important. Every group member must understand all steps. Simply knowing project goals is critical quality work. An extra ten minutes of planning can save an hour of work down the line. After all, its about working smarter, not harder. The same applies to assignment of roles, each member should have a role that fits their personality.

Gary: With respect to the actual work, starting early is something that cannot be overstated. You are going to run into problems, and being ahead of schedule simply gives you leeway to ask other group members for help or raise questions to the TA. Some of the problems may not be easy fixes; they may be problems deeply rooted in your code from previous erroneous implementations, so its important to leave yourself with enough time to prepare for the worst. You never want to feel pressured by the deadline. A happy groupy dynamic is healthy, which in turn makes the work flow much more efficient.

Mat: I would say that a major contributor to our group project's success is an understanding attitude. Sometimes, compromise can be difficult, and its hard to not let the frustration of compilation errors get to you. Working around schedule limitations of other group members helps ease the strain. In addition, it is important to not be stubborn about your own ideas.

Addendum: While referencing microC is a good starting point, do not to overly rely on it. We highly recommend starting by understand why the logic applies specifically to the features of that language. In addition, understand the LLVM module features in its entirety. By going through the llvm.moe documentation, we learned how LLVM actually worked under the hood. With this in mind, we tried our own approach and were able to quickly come up with our own working implementation.

8. Appendix

8.1. scanner.ml

Authors: Gary, Frank, Mat

```
(* Ocamllex scanner for PixCzar *)
```

```
{ open Parser }

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

let escapes = ’\\’[’\\’ ’"’ ’’’]

let stringcharacters = [’ ’ ’\t’ ’\r’ ’\n’ ’a’-’z’ ’A’-’Z’ ’0’-’9’
    ’!’ ’#’ ’%’ ’&’
’(’ ’)’ ’*’ ’+’ ’,’ ’-’ ’.’ ’/’ ’:’ ’;’ ’<’ ’=’ ’>’ ’?’ ’[’ ’]’
    ’^’ ’_’
’{’ ’|’ ’}’ ’~’] | escapes

rule token = parse
  [’ ’ ’\t’ ’\r’ ’\n’] { token lexbuf }
| "/*"    { comment lexbuf }
| "//"    { singlelinecomment lexbuf }
| ’(’     { LPAREN }
| ’)’     { RPAREN }
| ’{’     { LBRACE }
| ’}’     { RBRACE }
| ’[’     { LBRACK }
| ’]’     { RBRACK }
| ’;’     { SEMI }
| ’,’     { COMMA }
| ’+’     { PLUS }
| ’-’     { MINUS }
| ’*’     { TIMES }
| ’/’     { DIVIDE }
| ’=’     { ASSIGN }
| ’%’     { MOD }
| "=="    { EQ }
| "!="    { NEQ }
| ’<’     { LT }
| "<="    { LEQ }
| ">"     { GT }
| ">="    { GEQ }
| "&&"    { AND }
| "||"    { OR }
| "++"    { INCREMENT }
| "--"    { DECREMENT }
| "!"     { NOT }
| "if"    { IF }
| "else"  { ELSE }
| "else if" { ELSEIF }
| "for"   { FOR }
| "while" { WHILE }
| "break" { BREAK }
| "continue" { CONTINUE }
| "return" { RETURN }
| "Int"   { INT }
| "Boolean"{ BOOL }
| "Float" { FLOAT }
| "String" { STRING }
| "Void"  { VOID }
| "Pix"   { PIX }
| "Placement" { PLACEMENT }
| "Frame" { FRAME }
| "true"  { BLIT(true) }
| "false" { BLIT(false) }
| "null"  { NULL }
| "new"   { NEW }
| "."     { DOT }
| "~"     { TILDE }
| "#include"{ INCLUDE }
| digits as lxm { LITERAL(int_of_string lxm) }
| digits ’.’ digit* as lxm { FLIT(float_of_string lxm) }
| (’\"’ (stringcharacters* as lxm) ’\"’)|(’\’’ (stringcharacters*
   as lxm) ’\’’) { SLIT(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 }

and singlelinecomment = parse
```

```
  "\n" { token lexbuf }
| _    { singlelinecomment lexbuf }
```

8.2. parser.mly

Authors: Gary, Frank, Mat

```
%{open Ast%}

%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA PLUS MINUS TIMES
   DIVIDE ASSIGN MOD
%token NOT EQ NEQ LT LEQ GT GEQ AND OR INCREMENT DECREMENT TILDE
%token LBRACK RBRACK
%token RETURN IF ELSEIF ELSE FOR WHILE BREAK CONTINUE INCLUDE
%token INT BOOL FLOAT STRING VOID PIX PLACEMENT FRAME NULL NEW DOT
%token <int> LITERAL
%token <bool> BLIT
%token <string> ID SLIT
%token <float> FLIT
%token EOF

%start program
%type <Ast.program> program

%nonassoc NOELSE
%nonassoc ELSE
%nonassoc ELSEIF
%right ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE MOD
%right NOT NEG TILDE
%right PREINCREMENT PREDECREMENT
%left DOT INCREMENT DECREMENT

%%
```

```
program:
  decls EOF { $1 }

decls:
   /* nothing */ { ([], [])                              }
 | decls vdecl_list SEMI { (((List.rev $2) :: fst $1), snd $1) }
 | decls fdecl { (fst $1, ($2 :: snd $1))                }

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

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

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

prim_typ:
    INT   { Int   }
  | BOOL  { Bool  }
  | FLOAT { Float }
  | STRING { String }
  | VOID  { Void  }

nonprim_typ:
    PIX      { Pix     }
  | PLACEMENT { Placement }
  | FRAME    { Frame   }

typ:
    prim_typ        { $1          }
  | nonprim_typ     { $1          }
  | typ LBRACK RBRACK { Array($1, 0) }
```

```
vdecl_list:
    typ vdecl           { [(($1, snd (fst $2)), snd $2)]  }
  | vdecl_list COMMA vdecl { $3 :: $1                     }

vdecl:
    ID           { ((Notyp, $1), Noexpr) }
  | ID ASSIGN expr { ((Notyp, $1), $3) }

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

stmt:
    expr SEMI                             { Expr $1
                               }
  | RETURN expr_opt SEMI                  { Return $2
                           }
  | LBRACE stmt_list RBRACE               { Block(List.rev $2)
                  }
  | IF LPAREN expr RPAREN stmt %prec NOELSE
                                          { If($3, $5, Block([]),
                                             Block([])) }
  | IF LPAREN expr RPAREN stmt ELSE stmt
                                          { If($3, $5, Block([]),
                                             $7)      }
  | IF LPAREN expr RPAREN stmt elseif_list %prec NOELSE
                                          { If($3, $5,
                                             Block(List.rev $6),
                                             Block([])) }
  | IF LPAREN expr RPAREN stmt elseif_list ELSE stmt
                                          { If($3, $5,
                                             Block(List.rev $6),
                                             $8) }
  | FOR LPAREN expr_opt SEMI expr_opt SEMI expr_opt RPAREN stmt
                                          { For($3, $5, $7, $9)
                                                        }
  | WHILE LPAREN expr RPAREN stmt         { While($3, $5)
                       }
  | BREAK SEMI                            { Break
```

```
                               }
  | CONTINUE SEMI                         { Continue
                            }
  | INCLUDE SLIT SEMI                     { Include($2)
                         }
  | vdecl_list SEMI                       { VarDecs(List.rev $1)
                }
  | expr DOT ID LPAREN args_opt RPAREN SEMI { ObjCall($1, $3, $5)
              }

elseif_list:
  | ELSEIF LPAREN expr RPAREN stmt         { [ElseIf($3, $5)]   }
  | elseif_list ELSEIF LPAREN expr RPAREN stmt { ElseIf($4, $6) ::
     $1 }

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

expr:
    LITERAL        { Literal($1)          }
  | FLIT     { Fliteral($1)        }
  | BLIT           { BoolLit($1)          }
  | SLIT           { StringLit($1)        }
  | ID             { Id($1)               }
  | NULL           { NullLit              }
  | expr PLUS  expr { Binop($1, Add, $3)  }
  | expr MINUS expr { Binop($1, Sub, $3)  }
  | expr TIMES expr { Binop($1, Mult, $3) }
  | expr DIVIDE expr { Binop($1, Div, $3) }
  | expr MOD   expr { Binop($1, Mod, $3)  }
  | expr EQ    expr { Binop($1, Equal, $3) }
  | expr NEQ   expr { Binop($1, Neq, $3)  }
  | expr LT    expr { Binop($1, Less, $3) }
  | expr LEQ   expr { Binop($1, Leq, $3)  }
  | expr GT    expr { Binop($1, Greater, $3) }
  | expr GEQ   expr { Binop($1, Geq, $3)  }
  | expr AND   expr { Binop($1, And, $3)  }
  | expr OR    expr { Binop($1, Or,  $3)  }
  | MINUS expr %prec NEG { Unop(Neg, $2)  }
  | NOT expr       { Unop(Not, $2)        }
  | TILDE expr       { Unop(IntCast, $2)  }
  | expr ASSIGN expr { Assign($1, $3)     }
  | ID LPAREN args_opt RPAREN { Call($1, $3) }
  | LPAREN expr RPAREN { $2               }
  | NEW nonprim_typ LPAREN args_opt RPAREN { New($2, $4)
                 }
  | NEW nonprim_typ LBRACK LITERAL RBRACK { NewArray($2, $4)
             }
  | NEW prim_typ LBRACK LITERAL RBRACK { NewArray($2, $4)
                }
  | LBRACK args_opt RBRACK               { CreateArray($2)
                 }
  | ID LBRACK expr RBRACK                { AccessArray($1, $3)
             }
  | INCREMENT expr %prec PREINCREMENT  { Unop(PreIncrement, $2)
          }
  | DECREMENT expr %prec PREDECREMENT  { Unop(PreDecrement, $2)
          }
  | expr INCREMENT                     { PostUnop($1,
     PostIncrement) }
  | expr DECREMENT                     { PostUnop($1,
     PostDecrement) }

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

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

8.3. ast.ml

Authors: Mat, Frank, Bryan

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

```
type uop = Neg | Not | PreIncrement | PreDecrement | IntCast

type post_uop = PostIncrement | PostDecrement

type typ = Int | Bool | Float | String | Void | Pix | Placement |
    Frame | Notyp |
           Array of typ * int | Null

type expr =
    Literal of int
  | Fliteral of float
  | BoolLit of bool
  | StringLit of string
  | Id of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | PostUnop of expr * post_uop
  | Assign of expr * expr
  | Call of string * expr list
  | Noexpr
  | NullLit
  | New of typ * expr list
  | NewArray of typ * int
  | CreateArray of expr list
  | AccessArray of string * expr

type bind = typ * string

type var = bind * expr

type stmt =
    Block of stmt list
  | Expr of expr
  | Return of expr
  | If of expr * stmt * stmt * stmt
  | ElseIf of expr * stmt
  | For of expr * expr * expr * stmt
  | While of expr * stmt
  | Break
  | Continue
  | Include of string
  | VarDecs of var list
  | ObjCall of expr * string * expr list

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

type program = var list list * func_decl list

(* Pretty-printing functions *)

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

let string_of_uop = function
    Neg -> "-"
  | Not -> "!"
  | PreIncrement -> "++"
  | PreDecrement -> "--"
  | IntCast -> "~"

let string_of_post_uop = function
    PostIncrement-> "++"
  | PostDecrement-> "--"

let rec string_of_typ = function
    Int -> "Int"
  | Bool -> "Boolean"
  | Float -> "Float"
  | String -> "String"
  | Void -> "Void"
  | Pix -> "Pix"
  | Placement -> "Placement"
  | Frame -> "Frame"
  | Notyp -> ""
  | Array(t, _) -> string_of_typ t ^ "[]"
  | Null -> "Null"

let rec string_of_expr = function
    Literal(l) -> string_of_int l
  | Fliteral(l) -> string_of_float l
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | StringLit(l) -> "\"" ^ l ^ "\""
  | 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 ^ ")"
  | PostUnop(e, o) -> "(" ^ string_of_expr e ^ ")" ^
      string_of_post_uop o
  | Assign(e1, e2) -> string_of_expr e1 ^ " = " ^ string_of_expr e2
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^
          ")"
  | Noexpr -> ""
  | NullLit -> "null"
  | New(t, el) ->
     "new " ^ string_of_typ t ^ "(" ^ String.concat ", " (List.map
         string_of_expr el) ^ ")"
  | CreateArray(el) -> "[" ^ String.concat "," (List.map
      string_of_expr el) ^ "]"
  | AccessArray(id, e2) -> id ^ "[" ^ string_of_expr e2 ^ "]"
  | NewArray(t, i) -> "new " ^ string_of_typ t ^ "[" ^
     string_of_int i ^ "]"

let string_of_vdecl ((t, id), value) =
  match value with
  | Noexpr -> string_of_typ t ^ " " ^ id
  | _ -> string_of_typ t ^ " " ^ id ^ " = " ^ string_of_expr value

let string_of_vdecls (vars) = String.concat "," (List.map
   string_of_vdecl vars) ^
   if (List.length vars) > 0 then ";\n" else ""

let rec string_of_stmt = function
   Block(stmts) ->
       if (List.length stmts) > 0 then
       "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^
          "}\n" else ""
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
  | If(e, s1, s2, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^
  string_of_stmt s1 ^ string_of_stmt s2
  | If(e, s1, s2, s3) -> "if (" ^ string_of_expr e ^ ")\n" ^
     string_of_stmt s1 ^ string_of_stmt s2 ^ "else\n" ^
        string_of_stmt s3
  | ElseIf(e, s) -> "else if (" ^ string_of_expr e ^ ")\n" ^
     string_of_stmt s
  | 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
  | Break -> "break;\n"
  | Continue -> "continue;\n"
  | Include(s) -> "include " ^ s ^";\n"
  | VarDecs(vars) -> String.concat "," (List.map string_of_vdecl
     vars) ^ ";\n"
  | ObjCall(e, f, el) -> string_of_expr e ^ "." ^ f ^ "(" ^
     String.concat ", " (List.map string_of_expr el) ^ ");\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_stmt fdecl.body) ^
  "}\n"

let string_of_program (vars_list, funcs) =
  String.concat "" (List.map string_of_vdecls (List.rev vars_list))
      ^ "\n" ^
  String.concat "\n" (List.map string_of_fdecl (List.rev funcs))
```

8.4. sast.ml

Authors: Mat, Bryan, Frank

```
open Ast

module StringMap = Map.Make(String)

type sexpr = typ StringMap.t * typ * sx
and sx =
    SLiteral of int
  | SFliteral of float
  | SBoolLit of bool
  | SStringLit of string
  | SId of string
  | SBinop of sexpr * op * sexpr
  | SUnop of uop * sexpr
  | SPostUnop of sexpr * post_uop
  | SAssign of sexpr * sexpr
  | SCall of string * sexpr list
  | SNoexpr
  | SNullLit
  | SNew of typ * sexpr list
  | SNewArray of typ * int
  | SCreateArray of sexpr list
  | SAccessArray of string * sexpr
```

```
type svar = bind * sexpr

type sstmt = typ StringMap.t * ss
and ss =
    SBlock of sstmt list
  | SExpr of sexpr
  | SReturn of sexpr
  | SIf of sexpr * sstmt * sstmt * sstmt
  | SElseIf of sexpr * sstmt
  | SFor of sexpr * sexpr * sexpr * sstmt
  | SWhile of sexpr * sstmt
  | SBreak
  | SContinue
  | SInclude of string
  | SVarDecs of svar list
  | SObjCall of sexpr * string * sexpr list

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

type sprogram = svar list 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
  | SFliteral(l) -> string_of_float l
  | SBoolLit(true) -> "true"
  | SBoolLit(false) -> "false"
  | SStringLit(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
  | SPostUnop(e, o) -> string_of_sexpr e ^ string_of_post_uop o
  | SAssign(e1, e2) -> string_of_sexpr e1 ^ " = " ^ string_of_sexpr
     e2
  | SCall(f, el) ->
     f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^
         ")"
  | SNoexpr -> ""
  | SNullLit -> "null"
  | SNew(t, el) ->
     "new " ^ string_of_typ t ^ "(" ^ String.concat ", " (List.map
        string_of_sexpr el) ^ ")"
  | SCreateArray(el) -> "[" ^ String.concat "," (List.map
     string_of_sexpr el) ^ "]"
  | SAccessArray(id, e2) -> id ^ "[" ^ string_of_sexpr e2 ^ "]"
  | SNewArray(t, i) -> "new " ^ string_of_typ t ^ "[" ^
     string_of_int i ^ "]")

let string_of_svdecl ((t1, id), (map, t2, value)) =
  match value with
  | SNoexpr -> string_of_typ t1 ^ " " ^ id
  | _ -> string_of_typ t1 ^ " " ^ id ^ " = " ^ string_of_sexpr
     (StringMap.empty, t1, value)

let string_of_svdecls (vars) = String.concat "," (List.map
    string_of_svdecl vars) ^
    if (List.length vars) > 0 then ";\n" else ""

let rec string_of_sstmt (map, e) = match e with
    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, stmts, (map, SBlock([]))) -> "if (" ^ string_of_sexpr
     e ^ ")\n" ^ string_of_sstmt s
     ^ string_of_sstmt stmts
  | SIf(e, s1, stmts, s2) -> "if (" ^ string_of_sexpr e ^ ")\n" ^
     string_of_sstmt s1 ^ string_of_sstmt stmts ^ "else\n" ^
         string_of_sstmt s2
  | SElseIf(e, s) -> "else if (" ^ string_of_sexpr e ^ ")\n" ^
     string_of_sstmt s
  | 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
  | SBreak -> "break;\n"
  | SContinue -> "continue;\n"
  | SInclude(s) -> "include " ^ s ^ ";\n"
  | SVarDecs(vars) -> String.concat "," (List.map string_of_svdecl
     vars) ^ ";\n"
  | SObjCall(e, f, el) -> string_of_sexpr e ^ "." ^ f ^ "(" ^
     String.concat ", " (List.map string_of_sexpr el) ^ ");\n"

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_sstmt fdecl.sbody) ^
  "}\n"

let string_of_sprogram (vars_list, funcs) =
  String.concat "" (List.map string_of_svdecls (List.rev
     vars_list)) ^ "\n" ^
  String.concat "\n" (List.map string_of_sfdecl (List.rev funcs))
```

8.5. semant.ml

Authors: Gary, Bryan, Frank, Mat

```
open Ast
open Sast

module StringMap = Map.Make(String)

let check (globals, functions) =
  (* Check if a certain kind of binding has void type or is a
     duplicate
```

```
   of another, previously checked binding *)
let check_binds (kind : string) (to_check : bind list) =
  let check_it checked binding =
    let void_err = "illegal void " ^ kind ^ " " ^ snd binding
    and dup_err = "duplicate " ^ kind ^ " " ^ snd binding
    in match binding with
      (* No void bindings *)
      (Void, _) -> raise (Failure void_err)
    | (_, n1) -> match checked with
                  (* No duplicate bindings *)
                    ((_, n2) :: _) when n1 = n2 -> raise (Failure
                        dup_err)
                  | _ -> binding :: checked
  in let _ = List.fold_left check_it [] (List.sort compare
      to_check)
     in to_check
in let get_first lst = match lst with
    hd :: tl -> hd
  | _ -> ((Notyp, ""), Noexpr) in

let rec add_types typed_list var_list = (match var_list with
      ((_,s),e) :: tl -> let t = fst (fst (List.hd var_list)) in
        add_types (((t,s),e) :: typed_list) tl
    | _ -> typed_list) in

let get_binds var_list_list =
   let fold_types combine var_list =
       let var_list = List.rev (add_types [] var_list) in
       let rec fold_list combine var_list = (match var_list with
            ((t,s),_) :: tl -> fold_list ((t,s) :: combine) tl
          | _ -> combine)
       in fold_list combine var_list in
   List.fold_left fold_types [] var_list_list in

    (* Check vars to see if duplicate or void type *)
let check_vars (kind : string) (to_check: var list list) =
    let combined_list = get_binds to_check in
    let _ = ignore(check_binds kind combined_list)
  in to_check
in
```

```
let map_to_svar id typ resultlist = ((typ, id), (StringMap.empty,
    typ,
  SNoexpr)) :: resultlist in
let globals' = check_vars "global" globals in

(* Collect function declarations for built-in functions: no
    bodies *)
let built_in_decls =
  let add_bind map (name, typ, formal_vars) = StringMap.add name {
    typ = typ; fname = name;
    formals = formal_vars; locals = []; body = [] } map
  in List.fold_left add_bind StringMap.empty [("render", Int,
  [(Array(Frame,-1), "frames"); (Int, "fps"); (Int, "height");
      (Int, "width") ])]
in

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

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

(* Return a function from our symbol table *)
```

```
let find_func s =
  try StringMap.find s function_decls
  with Not_found -> raise (Failure ("unrecognized function " ^ s))
in

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

  (* Raise an exception if the given rvalue type cannot be
      assigned to
     the given lvalue type *)
  let check_assign lvaluet rvaluet err = match lvaluet with
      Pix | Placement | Frame ->
          if lvaluet = rvaluet || rvaluet = Null then lvaluet else
              raise (Failure err)
     | Array(t, _) -> (match rvaluet with
         Array(t, _) -> lvaluet
       | _ -> raise(Failure (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 (get_binds globals')
  in

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

  (* Return a semantically-checked expression, i.e., with a type
      *)
  let rec check_expr e map = match e with
      Literal  l -> (map, Int, SLiteral l)
    | Fliteral l -> (map, Float, SFliteral l)
    | BoolLit  l -> (map, Bool, SBoolLit l)
| StringLit l -> (map, String, SStringLit l)
| Noexpr     -> (map, Void, SNoexpr)
| NullLit    -> (map, Null, SNullLit)
| Id       s -> (map, type_of_identifier s map, SId s)
| Assign(le, e) as ex ->
   let (map, rt, e') = check_expr e map in
   let (map, t_le, le') = check_expr le map
   in let err = "illegal assignment " ^ string_of_typ t_le ^
       " = " ^
     string_of_typ rt ^ " in " ^ string_of_expr ex
   in let type_check = match le' with
       SId(s) -> (map, check_assign t_le rt err,
          SAssign((map, t_le, le'), (map, rt, e')))
     | SAccessArray(id, idx) ->
             (map, check_assign t_le rt err, SAssign((map,
                t_le, le'), (map, rt, e')))
     | _ -> raise (Failure(err)) in type_check
| Unop(op, e) as ex ->
   let (map, t, e') = check_expr e map in
   let ty = match op with
     Neg when t = Int || t = Float -> t
   | Not when t = Bool -> Bool
   | PreIncrement | PreDecrement when t = Int -> Int
   | IntCast when t = Float || t=Int -> Int
   | _ -> raise (Failure ("illegal unary operator " ^
                        string_of_uop op ^ string_of_typ t ^
                        " in " ^ string_of_expr ex))
   in (map, ty, SUnop(op, (map, t, e')))
| PostUnop(e, op) as ex ->
   let (map, t, e') = check_expr e map in
   let ty = match op with
     PostIncrement when t = Int -> t
   | PostDecrement when t = Int -> t
   | _ -> raise (Failure ("illegal postfix unary operator " ^
                        string_of_typ t ^ " in " ^
                           string_of_expr ex ^
                        string_of_post_uop op))
   in (map, ty, SPostUnop((map, t, e'), op))
| Binop(e1, op, e2) as e ->
   (* Determine expression type based on operator and
       operand types *)
   let (_, t1, e1') = check_expr e1 map
   and (_, t2, e2') = check_expr e2 map in
   let same = t1 = t2 in
   let ty = match op with
     Add when (same && t1 = String) || ((t1 = Int || t1 =
        Float) &&
         (t2 = Int || t2 = Float)) -> (match same with
              true -> t1
            | false -> Float)
   | Sub | Mult | Div when (t1 = Float || t1 = Int) &&
         (t2 = Float || t2 = Int) -> (match same with
              true -> t1
            | false -> Float)
   | Mod when same && t1 = Int -> Int
   | Equal | Neq when same -> Bool
   | Less | Leq | Greater | Geq
             when same && (t1 = Int || t1 = Float) -> Bool
   | And | Or when same && t1 = Bool -> Bool
   | _ -> raise (
  Failure ("illegal binary operator " ^
               string_of_typ t1 ^ " " ^ string_of_op op ^ " "
                  ^
               string_of_typ t2 ^ " in " ^ string_of_expr e))
   in (map, ty, SBinop((map, t1, e1'), op, (map, t2, e2')))
| Call(fname, args) as call -> (match fname with
   "print" -> if List.length args != 1 then
       raise (Failure ("expecting 1 argument in print"))
    else let (m, et, e') = check_expr (List.hd args) map in
        (match et with
        Int -> (map, Void, SCall("printi", [m, et, e']))
      | Float -> (map, Void, SCall("printf", [m, et, e']))
      | String -> (map, Void, SCall("prints", [m, et, e']))
      | Bool -> (map, Void, SCall("printb", [m, et, e']))
      | _ -> raise (Failure ("invalid argument for print")))
    | "length" -> if List.length args != 1 then
       raise (Failure ("expecting 1 argument in length"))
    else let (m, et, e') = check_expr (List.hd args) map in
        (match et with
      | Array(_,_) -> (map, Int, SCall("length", [m, et, e']))
```

```
       | _ -> raise (Failure ("invalid argument for length")))
   | _ ->
    let fd = find_func fname in
    let param_length = List.length fd.formals in
    if List.length args != param_length then
      raise (Failure ("expecting " ^ string_of_int
         param_length ^
                    " arguments in " ^ string_of_expr call))
    else let check_call (ft, _) e =
      let (map, et, e') = check_expr e map in
      let err = "illegal argument found " ^ string_of_typ et ^
               " expected " ^ string_of_typ ft ^ " in " ^
                  string_of_expr e
      in (map, check_assign et ft err, e')
    in
    let args' = List.map2 check_call fd.formals args
    in (map, fd.typ, SCall(fname, args')))
| New(t, args) as new_l ->
    let len_err len = "expecting " ^ string_of_int len ^
                    " arguments in " ^ string_of_expr new_l
    and check_arg ft e =
      let (map, et, e') = check_expr e map in
      let err = "illegal argument found " ^ string_of_typ et ^
               " expected " ^ string_of_typ ft ^ " in " ^
                  string_of_expr e
      in (map, check_assign ft et err, e')
    in let new_obj = match t with
        Pix      -> let check_pix args =
           if List.length args != 0 then raise (Failure
               (len_err 0)) else []
          in (map, Pix, SNew(Pix, check_pix args))
      | Placement -> let check_placement args =
           if List.length args != 3 then raise (Failure
               (len_err 3)) else
               List.map2 check_arg [Pix; Int; Int;] args
          in (map, Placement, SNew(Placement, check_placement
             args))
      | Frame    -> let check_frame args =
           if List.length args != 0 then raise (Failure
               (len_err 0)) else
             List.map2 check_arg [] args
         in (map, Frame, SNew(Frame, check_frame args))
      | _          -> raise (Failure ("illegal object name " ^
                string_of_typ t))
    in new_obj
| NewArray(s, size) as e ->
    let arr = match s with
        Int      -> (map, Array(Int, size), SNewArray(Int,
           size))
      | Float    -> (map, Array(Float, size), SNewArray(Float,
         size))
      | Bool     -> (map, Array(Bool, size), SNewArray(Bool,
         size))
      | String   -> (map, Array(String, size),
         SNewArray(String, size))
      | Pix      -> (map, Array(Pix, size), SNewArray(Pix,
         size))
      | Placement -> (map, Array(Placement, size),
         SNewArray(Placement, size))
      | Frame    -> (map, Array(Frame, size), SNewArray(Frame,
         size))
      | _          -> raise (Failure ("illegal array type in "
         ^

                        string_of_expr e))
  in if size > -1 then arr else
      raise (Failure ("illegal array size in " ^
         string_of_expr e))
| CreateArray(args) as e ->
    let err ext et = "illegal expression found " ^
       string_of_typ et ^
         "expected " ^ string_of_typ ext ^ " in " ^
             string_of_expr e
    in let check_types sexprs expr =
      let (map, et, e’) = check_expr expr map in
       match sexprs with
         (x, y, z) :: tl -> if (y) = et then (map, et, e’) ::
             sexprs else raise
                 (Failure (err (y) et))
         | _ -> (map, et, e’) :: sexprs
    in let result = List.fold_left check_types [] args in
```

```
      let arr = match result with
          (x,y,z) :: tl -> (map, Array(y, (List.length args)),
             SCreateArray(result))
        | _ -> (map, Array(Notyp, 0), SCreateArray(result))
      in arr
  | AccessArray(id, e2) ->
      let typ_err = id ^ " is not an array"
      in let (map2, et2, e2') = check_expr e2 map
      in let check_access = match (type_of_identifier id map)
          with
          (Array(typ, _)) -> (map2, typ, SAccessArray(id, (map2,
             et2, e2')))
        | _           -> raise (Failure (typ_err))
      in check_access
in

(* Return a semantically-checked statement i.e. containing
   sexprs *)
let rec check_stmt e map func loop_count = match e with
    Expr e -> (map, SExpr (check_expr e map))
  | If(e, s1, s2, s3) ->
      (map, SIf(check_expr e map, check_stmt s1 map func
          loop_count,
      check_stmt s2 map func loop_count, check_stmt s3 map func
          loop_count))
  | ElseIf(e, s) -> (map, SElseIf(check_expr e map, check_stmt
      s map func loop_count))
  | For(e1, e2, e3, st) -> let (x,y,z) = check_expr e1 map in
                         let (x', y',z') = check_expr e2 x in
                         let (x'',y'',z'') = check_expr e3 x' in
       (x'', SFor((x,y,z), (x',y',z'), (x'',y'',z''),
           check_stmt st x'' func
      (loop_count + 1)))
  | While(p, s) -> (map, SWhile(check_expr p map, check_stmt s
      map func
      (loop_count + 1)))
  | Return e -> let match_arrs t1 t2 = (match t1 with
       Array(ta1,_) -> (match t2 with
          Array(ta2,_) when ta1=ta2 -> true
         |_ -> false)
  | _ -> false) in
  let (map, t, e') = check_expr e map in
  if t = func.typ || match_arrs t func.typ then (map,
     SReturn (map, t, e'))
  else raise (
     Failure ("return gives " ^ string_of_typ t ^ " expected
        " ^
           string_of_typ func.typ ^ " in " ^ string_of_expr
              e))

 (* A block is correct if each statement is correct and
    nothing
   follows any Return statement. Nested blocks are
      flattened. *)
| Block sl ->
   let rec check_stmt_list e map = match e with
       [Return _ as s] -> [check_stmt s map func loop_count]
     | Return _ :: _ -> raise (Failure "nothing may follow a
        return")
     | Block sl :: ss -> check_stmt_list (sl @ ss) map (*
        Flatten blocks *)
     | s :: ss       -> let (x,y) = check_stmt s map func
        loop_count
         in (x,y) ::check_stmt_list ss x
     | []            -> []
   in (map, SBlock(check_stmt_list sl map))
| ObjCall(e, func, args) as s ->
   let err = "illegal object function call " ^
      string_of_stmt s in
   let check_func func_params args = if List.length args !=
      List.length func_params then raise (Failure(err)) else
        let check_call (ft, _) e =
            let (map, et, e') = check_expr e map
        in (map, check_assign ft et err, e')
      in List.map2 check_call func_params args
   in let checked_expr = check_expr e map
   in let check_rgb arr = let rgb = check_expr
     (List.hd arr) map in (match rgb with
        (_,Array(Int,3),_) -> rgb
       | _ -> raise(Failure("Invalid rgb input")))
   in let check_it = match checked_expr with
       (_, Pix, _) -> (match func with
           "makeRectangle" | "makeEllipse" -> let _ =
              check_rgb (List.rev args) in
             (map, SObjCall(checked_expr, func, check_func
                [(Int, "width"); (Int, "height");(Array(Int,
                   3),"rgb")]
                args))
          | "makeTriangle" -> let _ = check_rgb (List.rev
             args) in
             (map, SObjCall(checked_expr, func, check_func
                [(Int, "length");(Array(Int, 3),"rgb")] args))
          | "uploadImage" ->
             (map, SObjCall(checked_expr, func, check_func
              [(String, "path");(Int, "width");(Int,"height")]
                args))
          | "clear" -> (map, SObjCall(checked_expr, func,
             check_func []
            args))
          | _ -> raise(Failure("ObjCall not yet
             implemented")))
     | (_, Frame, _) -> (match func with
            "addPlacement" ->
             (map, SObjCall(checked_expr, func, check_func
                [(Placement, "place")] args))
          | "clearPlacements" ->
             (map, SObjCall(checked_expr, func, check_func []
                args))
          | _ -> raise(Failure("ObjCall not yet
             implemented")))
     | _ -> raise (Failure(err))
   in check_it
| VarDecs(field) -> let t = fst (fst (get_first field)) in
  let vardecs (map, vardecs_list) (b, e) =
  let s = snd b
  in let _ = (if t=Void then raise(Failure("Void type
      declaration")))
  in (if StringMap.mem s map then raise ( Failure
      ("Duplicate variable declaration " ^ s))
      else let (map, t2, _) = check_expr e map in
```

```
                let new_symbols = if t2=Void then StringMap.add s
                    t map else
                    StringMap.add s t2 map
                   in let err = "LHS type of " ^ string_of_typ t ^
                       " not the same as " ^
                     "RHS type of " ^ string_of_typ t2 in
                   let _ = if t2 <> Void then check_assign t t2
                       err else t in
            (new_symbols, (b, check_expr e new_symbols) ::
                vardecs_list)) in
        let (new_symbols, vardecs_list) = List.fold_left vardecs
            (map, []) field in
           (new_symbols, SVarDecs(List.rev vardecs_list))

  | Continue -> if loop_count > 0 then (map, SContinue) else
      raise(Failure("Continue statement not in loop"))
  | Break -> if loop_count > 0 then (map, SBreak) else
      raise(Failure("Break statement not in loop"))
  | _ -> raise (Failure("To implement statement"))

  in let check_function func =
 let formals' = check_binds "formal" func.formals in
 let symbols_with_formals = List.fold_left (fun m (ty, name) ->
     StringMap.add name ty m)
 symbols func.formals in
 let sbody_stmt = check_stmt (Block func.body)
     symbols_with_formals func 0

 in (* body of check_function *)
 { styp = func.typ;
   sfname = func.fname;
   sformals = formals';
   slocals = StringMap.fold map_to_svar (fst sbody_stmt) [];
   sbody = match sbody_stmt with
(_ , SBlock(sl)) -> sl
  | _ -> let err = "internal error: block didn't become a
      block?"
   in raise (Failure err)
 }
```

```
    in let global_var_check svar_list var_list =
      let t = fst (fst (get_first var_list)) in
      let _ = if t<>Int&&t<>Float&&t<>String&&t<>Bool
        then raise(Failure("Invalid type for global variable")) in
      let vardecs vardecs_list ((_,s), e) =
        let (map2,t2,e2) = check_expr e StringMap.empty
        in let err = "LHS type of " ^ string_of_typ t ^ " not the
            same as " ^
        "RHS type of " ^ string_of_typ t2 in
        let _ = (if t2 <> Void then check_assign t t2 err else t)
        in ((t,s), (map2,t2,e2)) :: vardecs_list
      in let vardecs_list = List.rev (List.fold_left vardecs []
          var_list)
      in vardecs_list :: svar_list

    in let globals' = List.fold_left global_var_check [] globals'
    in let rec move_main lst el = (match el with
        hd :: tl -> move_main (if hd.sfname="main" then lst@[hd]
            else hd::lst) tl
      | _ -> lst)
    in let functions =
        {styp=Void;sfname="check_access";sformals=[(Int,"idx");(Int,"size")];slocals=[];s
            ::
            (List.map check_function functions)
    in let functions = List.rev (move_main [] functions)
    in (globals', functions)
```

8.6. codegen.ml

Authors: Frank, Bryan, Gary, Mat

```
module L = Llvm
module A = Ast
open Sast

module StringMap = Map.Make(String)
module Hash = Hashtbl

let local_values:(string, L.llvalue) Hash.t = Hash.create 50
let global_values:(string, L.llvalue) Hash.t = Hash.create 50
let array_info:(string, int) Hash.t = Hash.create 50

(* Code Generation from the SAST. Returns an LLVM module if
    successful,
   throws an exception if something is wrong. *)
let translate (globals, functions) =
  let context  = L.global_context () in
  (* Add types to the context so we can use them in our LLVM code *)
  let i32_t     = L.i32_type   context
  and i8_t      = L.i8_type    context
  and void_t    = L.void_type  context
  and str_t     = L.pointer_type (L.i8_type context)
  and float_t   = L.double_type context
  and i1_t      = L.i1_type    context in

  let pix_struct  = L.named_struct_type context "pix" in
  let () = L.struct_set_body pix_struct [|i32_t; str_t; i32_t;
      i32_t; L.pointer_type
  i32_t|] false in
  let pix_t = L.pointer_type pix_struct in

  let placement_struct = L.named_struct_type context "placement" in
    let () = L.struct_set_body placement_struct
      [| pix_t; i32_t; i32_t |] false in
  let placement_t = L.pointer_type placement_struct in
  let placement_node = L.named_struct_type context "placement_node"
      in
  let placement_node_t = L.pointer_type placement_node in
    let () = L.struct_set_body placement_node
      [| placement_node_t; placement_t |] false in

  let frame_struct = L.named_struct_type context "frame" in
    let () = L.struct_set_body frame_struct [|placement_node_t;|]
        false in
  let frame_t = L.pointer_type frame_struct in

  let the_module = L.create_module context "PixCzar" in

  let rec ltype_of_typ = function
      A.Int      -> i32_t
    | A.Void     -> void_t
    | A.String   -> str_t
    | A.Float    -> float_t
    | A.Bool     -> i1_t
    | A.Null     -> i32_t
    | A.Array(t, _) -> L.pointer_type (ltype_of_typ t)
    | A.Pix      -> pix_t
    | A.Placement -> placement_t
    | A.Frame    -> frame_t
    | t -> raise (Failure ("Type " ^ A.string_of_typ t ^ " not
       implemented yet"))
  in

  (* declare built-in functions *)
  let builtin_exit_t : L.lltype =
      L.var_arg_function_type void_t [||] in
  let builtin_exit_func : L.llvalue =
     L.declare_function "exit" builtin_exit_t the_module in
  let builtin_printf_t : L.lltype =
      L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
  let builtin_printf_func : L.llvalue =
     L.declare_function "printf" builtin_printf_t the_module in
  let builtin_render_t : L.lltype =
      L.var_arg_function_type i32_t [| i32_t; L.pointer_type
         frame_t; i32_t; i32_t;
      i32_t; |] in
  let builtin_render_func : L.llvalue =
     L.declare_function "render" builtin_render_t the_module in

  let to_imp str = raise (Failure ("Not yet implemented: " ^ str))
      in

  (* Define each function (arguments and return type) so we can
   * define it's body and call it later *)
  let function_decls : (L.llvalue * sfunc_decl) StringMap.t =
    let function_decl m fdecl =
      let name = fdecl.sfname
      and formal_types =
   Array.of_list (List.map (fun (t,_) -> ltype_of_typ t)
      fdecl.sformals)
      in let ftype = L.function_type (ltype_of_typ fdecl.styp)
         formal_types in
      StringMap.add name (L.define_function name ftype the_module,
         fdecl) m in
    List.fold_left function_decl StringMap.empty functions in

    let int_format_str builder = L.build_global_stringptr "%d\n"
        "fmt" builder in
    let string_format_str builder = L.build_global_stringptr "%s\n"
        "fmt" builder in
    let float_format_str builder = L.build_global_stringptr "%g\n"
        "fmt" builder in
    let bool_format_str builder = L.build_global_stringptr "%d\n"
        "fmt" builder in

    let fill_struct structobj el builder =
       let store_el idx e =
          let e_p = L.build_struct_gep structobj idx
              "struct_build" builder
          in ignore(L.build_store e e_p builder)
       in List.iteri store_el el in

    let typ_malloc typ_ptr typ_struct el_arr builder =
       let struct_malloc = L.build_malloc typ_struct "malloc"
          builder in
       let struct_malloc = L.build_pointercast struct_malloc
          typ_ptr "cast"
         builder in
       let () = fill_struct struct_malloc el_arr builder
       in struct_malloc in

    let create_array_gen builder lt size =
      let size_arr = L.const_int i32_t (size+1) in
      let arr = L.build_array_malloc lt size_arr "array_gen"
         builder in
      let arr = L.build_pointercast arr (L.pointer_type lt)
         "array_cast"
      builder in
      let casted = L.build_gep arr [|L.const_int i32_t 0|] "size"
        builder in
      let casted = L.build_pointercast casted (L.pointer_type
         i32_t) "cast"
      builder in
      let _ = ignore(L.build_store (L.const_int i32_t size)
      casted builder) in arr in

    let rec gen_default_value t builder = let zero = L.const_int
        i32_t 0 in
    match t with
        A.Int -> L.const_int i32_t 0
      | A.Float -> L.const_float float_t 0.0
      | A.Bool -> L.const_int i1_t 0
      | A.String -> L.build_global_stringptr "" "tmp" builder
      | A.Pix -> typ_malloc pix_t pix_struct
          [zero;L.const_pointer_null str_t;zero;zero;
          L.const_pointer_null (L.pointer_type i32_t)] builder
      | A.Placement -> let pix = gen_default_value A.Pix builder in
          typ_malloc placement_t placement_struct [pix; zero;zero]
          builder
      | A.Frame -> let node = typ_malloc placement_node_t
         placement_node
          [L.const_pointer_null placement_node_t;
              L.const_pointer_null
            placement_t] builder in
          typ_malloc frame_t frame_struct [node] builder
      | A.Array(typ,_) -> create_array_gen builder (ltype_of_typ
         typ) 0
      | _ -> raise(Failure("No default value for this type")) in

    let rec expr builder ((m, t, e) : sexpr) = match e with
        SLiteral i -> L.const_int i32_t i
      | SStringLit st -> L.build_global_stringptr st "tmp" builder
      | SFliteral l -> L.const_float float_t l
      | SBoolLit b -> L.const_int i1_t (if b then 1 else 0)
      | SNoexpr -> L.const_int i32_t 0
      | SId s -> id_gen builder s true
      | SAssign(e1, e2) -> assign_gen builder e1 e2
      | SCall (id, e) ->
         let build_expr_list expr_list e = (expr builder e) ::
            expr_list in
```

```
(match id with
  "printf" ->
    L.build_call builtin_printf_func [| float_format_str
        builder ; (expr
       builder (List.hd e)) |] "printf" builder
| "printi" ->
    L.build_call builtin_printf_func [| int_format_str
        builder ; (expr builder
       (List.hd e)) |] "printf" builder
| "prints" ->
    L.build_call builtin_printf_func [| string_format_str
        builder ; (expr builder
       (List.hd e)) |] "printf" builder
| "printb" ->
    L.build_call builtin_printf_func [| bool_format_str
        builder ; (expr builder
       (List.hd e)) |] "printf" builder
| "length" -> let (_,t,ss) = List.hd e in (match ss with
    SId(s) -> (match t with
       Array(t,_) -> let arr = id_gen builder s true in
         let ptr = L.build_gep arr [|L.const_int i32_t 0|]
             "size" builder in
         let size = L.build_bitcast ptr (L.pointer_type
             i32_t) "cast"
           builder in
         L.build_load size "size" builder
     | _ -> raise(Failure("Incorrect length input")))
   | SNewArray(_,idx) -> L.const_int i32_t idx
   | SCreateArray(el) -> L.const_int i32_t (List.length el)
   | _ -> raise(Failure("incorrect type")))
| "render" -> let size = (match List.hd e with
   | (_,A.Array(_,size),_) -> size
   | _ -> raise(Failure("Invalid render input"))) in
    let arg_list = (List.rev (List.fold_left build_expr_list
      [L.const_int i32_t (size)] e)) in
    L.build_call builtin_render_func (Array.of_list
        arg_list) "render" builder
| _ -> if StringMap.mem id function_decls then
         let (the_function, fdecl) = StringMap.find id
             function_decls in
           let arg_list = Array.of_list
             (List.rev(List.fold_left build_expr_list [] e))
                in
           if fdecl.styp = Void then
             L.build_call the_function arg_list "" builder
                else
             L.build_call the_function arg_list id builder
         else raise(Failure("Built-in function not
             implemented"))
  )
| SBinop (e1, op, e2) -> binop_gen builder e1 op e2
| SUnop(op, e) -> unop_gen builder op e
| SNullLit -> L.const_null i32_t
| SNew(t, el) -> let rec to_ll ll_list el_list = (match
   el_list with
   hd :: tl -> to_ll ((expr builder hd) :: ll_list) tl
  | _ -> List.rev(ll_list)) in let arr = to_ll [] el in
   (match t with (*ToDo: garbage collection*)
   Pix -> gen_default_value A.Pix builder
  | Placement -> typ_malloc placement_t placement_struct arr
     builder
  | Frame -> gen_default_value A.Frame builder
  | _ -> to_imp "Additional types")
| SNewArray(t, size) -> let lt = ltype_of_typ t in
  let arr = create_array_gen builder lt size in
  let rec fill size = (match size with
    0 -> ()
   | _ -> let _ = ignore(L.build_store (gen_default_value t
      builder)
    (L.build_gep arr [| (L.const_int i32_t size) |]
        "array_assign"
    builder) builder) in fill (size-1)) in
    let _ = fill (size) in arr
| SCreateArray(el) -> let e = List.hd el in let (_, t, _) = e
   in
   let lt = ltype_of_typ t in
   let arr = create_array_gen builder lt (List.length el) in
   let _ = fill_array builder arr (List.rev el) in arr
| SAccessArray(name, idx) -> access_array_gen builder name
   idx false
  | SPostUnop(e, op) -> let (_, _, e'') = e in let e' = expr
     builder e in (match op with
       PostIncrement -> (match e'' with
         SId(_) | SAccessArray(_, _) ->
           assign_gen builder e (m, t, SBinop(e, A.Add, (m, t,
              SLiteral(1))))
         | _ -> L.build_add e' (L.const_int i32_t 1) "add"
            builder)
     | PostDecrement -> (match e'' with
         SId(_) | SAccessArray(_, _) ->
           assign_gen builder e (m, t, SBinop(e, A.Sub, (m, t,
              SLiteral(1))))
         | _ -> L.build_sub e' (L.const_int i32_t 1) "sub"
            builder)
  )

and id_gen builder id deref =
    if Hash.mem local_values id then
       let _val = Hash.find local_values id in
       if deref = true then
         L.build_load _val id builder
       else _val
    else if Hash.mem global_values id then
       let _val = Hash.find global_values id in
       if deref = true then
         L.build_load _val id builder
       else _val
    else
     raise(Failure("Unknown variable: " ^ id))

and assign_gen builder se1 se2 =
  let (_, t1, e1) = se1 in
  let (_, t2, e2) = se2 in

  let rhs = (match e2 with
     SId(id) -> id_gen builder id true
    | SAccessArray(name, idx) -> access_array_gen builder name
       idx true
    | _ -> expr builder se2) in
  let rhs = (match t2 with
      A.Null -> L.const_null (ltype_of_typ t2)
    | _ -> rhs) in
  let _ = (match e1 with
      SId id -> let _ = (match e2 with
         SCreateArray(el) -> ignore(Hash.add array_info id
             (List.length el))
       | SNewArray(_,s) -> ignore(Hash.add array_info id s)
       | _ -> ()) in let lhs = id_gen builder id false in
       ignore(L.build_store rhs lhs builder)
    | SAccessArray(name, idx) -> let lhs = access_array_gen
       builder name idx
       true in ignore(L.build_store rhs lhs builder)
    | _ -> raise(Failure("Unable to assign " ^ string_of_sexpr
       se1 ^ " to "
                       ^ string_of_sexpr se2))) in
  rhs

and fill_array builder arr el =
  let array_assign idx arr_e = ignore(L.build_store (expr
      builder
  arr_e)
  (L.build_gep arr [| (L.const_int i32_t (idx+1)) |]
      "array_assign"
    builder) builder) in
  List.iteri array_assign el

and llvm_int_to_int llint =
  let llint = L.int64_of_const llint in match llint with
      Some(x) -> Int64.to_int(x)
    | _ -> raise(Failure("int64 operation failed"))

and access_array_gen builder name index is_assign =
  let (_,_,ss) = index in
  let arr = id_gen builder name true in
  let index = expr builder index in
  let _ = (match ss with
     SLiteral(i) ->
      let int_index = llvm_int_to_int index in
      let int_size = Hash.find array_info name in
      (if int_size > -1 && (int_index < 0 || int_index >=
          int_size)
        then raise(Failure("Illegal index")))
    | _ -> ()) in
  let (check,_) = StringMap.find "check_access" function_decls
      in
  let size = L.build_gep arr [| L.const_int i32_t 0 |] "size"
      builder in
  let size = L.build_pointercast size (L.pointer_type i32_t)
      "cast" builder in
  let size = L.build_load size "size" builder in
  let _ = L.build_call check [|index; size|] "" builder in
  let index = L.build_add index (L.const_int i32_t 1) "add"
      builder in
  let arr_val = L.build_gep arr [| index |] "arr_access"
      builder in
  if is_assign then arr_val
  else L.build_load arr_val "arr_access_val" builder

and f_op op = match op with
    A.Add    -> L.build_fadd
  | A.Sub    -> L.build_fsub
  | A.Mult   -> L.build_fmul
  | A.Div    -> L.build_fdiv
  | A.Equal  -> L.build_fcmp L.Fcmp.Oeq
  | A.Neq    -> L.build_fcmp L.Fcmp.One
  | A.Less   -> L.build_fcmp L.Fcmp.Olt
  | A.Leq    -> L.build_fcmp L.Fcmp.Ole
  | A.Greater -> L.build_fcmp L.Fcmp.Ogt
  | A.Geq    -> L.build_fcmp L.Fcmp.Oge
  | A.And | A.Or | A.Mod ->
      raise (Failure "internal error: semant should have
          rejected and/or on float")

and binop_gen builder e1 op e2 =
  let (_, t1, _) = e1 and (_, t2, _) = e2
    and e1' = expr builder e1
    and e2' = expr builder e2 in
    if t1=t2 && t1 = A.Float then (f_op op) e1' e2' "tmp"
        builder
    else if t1=t2 then (match op with
    | A.Add    -> L.build_add
    | A.Sub    -> L.build_sub
    | A.Mult   -> L.build_mul
    | A.Div    -> L.build_sdiv
    | A.Mod    -> L.build_srem
    | A.And    -> L.build_and
    | A.Or     -> L.build_or
    | A.Equal  -> L.build_icmp L.Icmp.Eq
    | A.Neq    -> L.build_icmp L.Icmp.Ne
    | A.Less   -> L.build_icmp L.Icmp.Slt
    | A.Leq    -> L.build_icmp L.Icmp.Sle
    | A.Greater -> L.build_icmp L.Icmp.Sgt
    | A.Geq    -> L.build_icmp L.Icmp.Sge
    ) e1’ e2’ "tmp" builder
 else
    (f_op op) (if t1=Int then L.build_sitofp e1’ float_t
       "cast" builder else e1’)
    (if t2=Int then L.build_sitofp e2’ float_t "cast" builder
       else e2’) "tmp" builder

and unop_gen builder unop e =
let unop_lval = expr builder e
and (m, t, e’) = e in match unop, t with
    A.Neg, A.Int -> L.build_neg unop_lval "neg_int_tmp"
       builder
  | A.Neg, A.Float -> L.build_fneg unop_lval "neg_flt_tmp"
     builder
  | A.Not, A.Bool -> L.build_not unop_lval "not_bool_tmp"
     builder
  | A.PreIncrement, _ -> (match e’ with
      SId(_) | SAccessArray(_, _) -> let _ = assign_gen
          builder e (m, t, SBinop(e,
         A.Add, (m, t, SLiteral(1)))) in unop_lval
    | _ -> let _ = L.build_add unop_lval (L.const_int i32_t
       1) "add" builder
      in unop_lval)
  | A.PreDecrement, _ -> (match e’ with
      SId(_) | SAccessArray(_, _) -> let _ = assign_gen
          builder e (m, t, SBinop(e,
         A.Sub, (m, t, SLiteral(1)))) in unop_lval
```

```
        | _ -> let _ = L.build_sub unop_lval (L.const_int i32_t
            1) "add" builder
           in unop_lval)
      | A.IntCast, A.Float -> L.build_fptosi unop_lval i32_t
          "cast" builder

      | A.IntCast, A.Int -> unop_lval
      | _ -> raise(Failure("Unsupported unop for " ^
          A.string_of_uop unop ^
        " and type " ^ A.string_of_typ t))
      in

 let build_vars svar_list hashtable builder =
     let ((t,_),_) = List.hd svar_list in
     let build_var builder svar =
         let ((_, s), (m, et, e')) = svar in
         let svar' = if et=A.Void then
             gen_default_value t builder else
             expr builder (m, et, e') in
         let lltype = ltype_of_typ t in
         let alloca = L.build_alloca lltype s builder in
         let _ = (match t with
             Array(_,size) -> Hash.add array_info s (match et with
             Array(_,et_size) -> et_size
               | _ -> 0)
             | _ -> ()) in
         let _ = Hash.add hashtable s alloca in
         let _ = ignore(L.build_store svar' alloca builder) in
             builder
      in List.fold_left build_var builder svar_list in

let build_global svar_list =
    let build_svar svar =
     let ((t, s), _) = svar in
     let lltype = ltype_of_typ t in
     ignore(Hash.add global_values s (L.declare_global lltype s
         the_module)) in
    List.iter build_svar svar_list in

 let _ = List.iter build_global globals in
let bool_pred_gen builder e = let (_,t,sx) = e in
  let e' = expr builder e in (match t with
      A.Bool -> e'
    | A.Int -> L.build_icmp L.Icmp.Sgt e' (L.const_int i32_t 0)
       "tmp" builder
    | A.Float -> L.build_fcmp L.Fcmp.Ogt e' (L.const_float
       float_t 0.0) "tmp" builder
    | _ -> (match sx with
       SNullLit -> L.const_int i1_t 0
      | _ -> L.const_int i1_t 1)) in

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

  let _ = (if fdecl.sfname="main" then
      let declare_globals svar_list =
       let ((t,_),_) = List.hd svar_list in
       let declare_svar svar =
       let ((_, s), (m', t',e')) = svar in
       let svar' = if t'=A.Void then
         gen_default_value t builder else expr builder (m',
             t',e') in
       ignore(Hash.add global_values s (L.define_global s svar'
           the_module))
       in List.iter declare_svar svar_list
      in List.iter declare_globals globals) in

   let func_params idx (t,s) = let _val = L.param the_function
      idx in
    let alloca = L.build_alloca (L.type_of _val) "param_alloc"
       builder in
    let _ = ignore(L.build_store _val alloca builder) in
    let _ = (match t with
      A.Array(_,size) -> ignore(Hash.add array_info s
          (Int32.to_int(Int32.max_int)))
      | _ -> ()) in
    ignore(Hash.add local_values s alloca) in
  let _ = List.iteri func_params fdecl.sformals in

  let _ = (if fdecl.sfname="check_access" then
   let checked_block = L.append_block context "checked"
       the_function in
   let _ = L.build_ret_void (L.builder_at_end context
       checked_block) in
   let exit_block = L.append_block context "exit" the_function in
   let exit_block_b = L.builder_at_end context exit_block in
   let str = L.build_global_stringptr "Throwing Runtime Error:
       Out of Bound Array Access" "tmp" builder in
   let _ = L.build_call builtin_printf_func [| string_format_str
       builder ;
       str|] "printf" exit_block_b in
   let _ = L.build_call builtin_exit_func [||] "" exit_block_b in
   let _ = L.build_ret_void exit_block_b in

   let idx = L.param the_function 0 in
   let size = L.param the_function 1 in
   let cmp = L.build_icmp L.Icmp.Sgt size idx "tmp" builder in
   ignore(L.build_cond_br cmp checked_block exit_block builder))
       in

(* Each basic block in a program ends with a "terminator"
    instruction i.e.
one that ends the basic block. By definition, these
    instructions must
indicate which basic block comes next -- they typically yield
    "void" value
and produce control flow, not values *)
(* Invoke "instr builder" if the current block doesn't already
   have a terminator (e.g., a branch). *)
let add_terminal builder instr =
                     (* The current block where we're inserting
                        instr *)
  match L.block_terminator (L.insertion_block builder) with
    Some _ -> ()
  | None -> (match instr with
     Some(instr) -> ignore (instr builder)
    | None -> raise(Failure("No default return value for this
       type"))) in

let rec stmt builder (map, ss) loop_list = match ss with
    SExpr e -> let _ = expr builder e in builder
  | SBlock sl -> let stmt_block builder s = stmt builder s
     loop_list in
     List.fold_left stmt_block builder sl
  | SReturn e -> let _ = match fdecl.styp with
     A.Void -> L.build_ret_void builder
    | _ -> L.build_ret (expr builder e) builder
    in builder
  | SWhile (predicate, body) ->
     (* First create basic block for condition instructions --
        this will
     serve as destination in the case of a loop *)
    let pred_bb = L.append_block context "while" the_function in
         (* In current block, branch to predicate to execute
             the condition *)
    let _ = L.build_br pred_bb builder in
    let merge_bb = L.append_block context "merge" the_function
       in

         (* Create the body’s block, generate the code for it,
             and add a branch
         back to the predicate block (we always jump back at
             the end of a while
         loop’s body, unless we returned or something) *)
    let body_bb = L.append_block context "while_body"
       the_function in
     let while_builder = stmt (L.builder_at_end context
        body_bb) body
         ((pred_bb, merge_bb) :: loop_list) in
     let () = add_terminal while_builder (Some(L.build_br
        pred_bb)) in
         (* Generate the predicate code in the predicate block
             *)
    let pred_builder = L.builder_at_end context pred_bb in
    let bool_val = bool_pred_gen pred_builder predicate in
  let _ = L.build_cond_br bool_val body_bb merge_bb
      pred_builder in
  L.builder_at_end context merge_bb

| SFor (e1, e2, e3, body) -> stmt builder
 (map, ( SBlock [(map, SExpr e1) ; (map, SWhile (e2, (map,
     SBlock [body ;
      (map, SExpr e3)]))) ])) loop_list
| SVarDecs(svar_list) ->
      build_vars svar_list local_values builder
| SIf (predicate, then_stmt, elseif_stmts, else_stmt) ->
        if_gen builder predicate then_stmt elseif_stmts
            else_stmt loop_list
| SElseIf (_, _) -> raise(Failure("Should never reach
    SElseIf"))
| SBreak -> let () = add_terminal builder (Some(L.build_br
    (snd (List.hd
loop_list)))) in builder
| SContinue -> let () = add_terminal builder (Some
    (L.build_br (fst (List.hd
loop_list)))) in builder
| SObjCall(e, name, el) -> let build_expr_list expr_list e =
    (expr builder e) :: expr_list in (match name with
    "addPlacement" -> let frame = expr builder e in
       let placement = expr builder (List.hd el) in
       let pnode_ptr = L.build_struct_gep frame 0 "add_plcmt"
           builder in
       let prev_node = L.build_load pnode_ptr "node" builder in
       let node = typ_malloc placement_node_t placement_node
         [prev_node; placement] builder in
       let _ = ignore(L.build_store node pnode_ptr builder) in
       builder
   | "clearPlacements" -> let frame = expr builder e in
       let node_ptr = L.build_struct_gep frame 0
           "clear_plcmts" builder in
       let _ = ignore(L.build_store (L.const_pointer_null
       placement_node_t) node_ptr builder) in
       let plcmt_ptr = L.build_struct_gep frame 1
           "clear_plcmts" builder in
        let _ = ignore(L.build_store (L.const_pointer_null
        placement_t) plcmt_ptr builder) in builder
     | "makeRectangle" -> let pix = expr builder e in
        let _ = fill_struct pix
        (List.rev(List.fold_left build_expr_list
        [L.const_pointer_null str_t; L.const_int i32_t 1] el))
            builder
        in builder
     | "makeTriangle" -> let pix = expr builder e in
        let _ = fill_struct pix
        (List.rev(List.fold_left build_expr_list [L.const_int
            i32_t 0;
        L.const_pointer_null str_t; L.const_int i32_t 2]
          el)) builder
        in builder
     | "makeEllipse" -> let pix = expr builder e in
        let _ = fill_struct pix
        (List.rev(List.fold_left build_expr_list
        [L.const_pointer_null str_t; L.const_int i32_t 3] el))
            builder
        in builder
     | "uploadImage" -> let pix = expr builder e in
        let _ = fill_struct pix (List.rev(List.fold_left
        build_expr_list [L.const_int i32_t 4] el)) builder in
            builder
     | "clear"-> let pix = expr builder e in
        let _ = fill_struct pix (List.rev(List.fold_left
        build_expr_list [L.const_int i32_t 0] el)) builder in
            builder
     | _ -> let _ = to_imp "object call: " ^ name in builder
  )
  | s -> to_imp (string_of_sstmt (map, ss))

and if_gen builder predicate then_stmt elseif_stmts else_stmt
    loop_list =
  let if_bool_val = bool_pred_gen builder predicate in
  let if_bb = L.append_block context "if" the_function in
  let () = add_terminal builder (Some (L.build_br if_bb)) in

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

  let then_bb = L.append_block context "then" the_function in
  let then_builder = stmt (L.builder_at_end context then_bb)
      then_stmt
    loop_list in
  let () = add_terminal then_builder (Some branch_instr) in

  let else_bb = L.append_block context "else" the_function in
  let else_builder = stmt (L.builder_at_end context else_bb)
      else_stmt
    loop_list in
  let () = add_terminal else_builder (Some branch_instr) in

  let rec elseif_bb_gen elseif_list bool_val pred_bb body_bb
      loop_list = match elseif_list with
       (_, SElseIf(pred, body)) :: tl ->
        let elseif_pred_val = expr (L.builder_at_end context
            pred_bb) pred in
        let elseif_pred_bb = L.append_block context
            "elseif_pred" the_function in

        let elseif_body_bb = L.append_block context
            "elseif_body" the_function in
        let elseif_builder = stmt (L.builder_at_end context
            elseif_body_bb)
          body loop_list in
        let () = add_terminal elseif_builder (Some
            branch_instr) in

        let _ = L.build_cond_br bool_val body_bb elseif_pred_bb
           (L.builder_at_end context pred_bb)
        in elseif_bb_gen tl elseif_pred_val elseif_pred_bb
            elseif_body_bb loop_list

     | _ -> L.build_cond_br bool_val body_bb else_bb
           (L.builder_at_end context pred_bb)
   in let elseif_ss = match elseif_stmts with
       (_, SBlock(elseif_ss)) -> elseif_ss
     | _ -> raise(Failure("Elseif must contain a Block"))
```

```
      in let _ = elseif_bb_gen elseif_ss if_bool_val if_bb then_bb
          loop_list in
      L.builder_at_end context merge_bb
    in
      (* Build the code for each statement in the function *)
    let builder = stmt builder (StringMap.empty, SBlock
        fdecl.sbody) [] in

    (* Add a return if the last block falls off the end *)
    add_terminal builder (match fdecl.styp with
        A.Void -> (Some L.build_ret_void)
      | A.Float -> (Some (L.build_ret (L.const_float float_t 0.0)))
      | A.Int -> (Some (L.build_ret (L.const_int i32_t 0)))
      | A.Bool -> (Some (L.build_ret (L.const_int i1_t 0)))
      | t -> None)

    in List.iter build_function_body functions; the_module
```

8.7. main.c

Author: Frank

```
#define GLEW_STATIC

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <GL/glew.h>
#include <GLFW/glfw3.h>
#include "soil.h"

struct pix {
    int type;
    char *filename;
    int width;
    int height;
    int *rgb;
} typedef pix;
```

```
struct placement {
    pix *ref;
    int x;
    int y;
    int rank;
    int group;
} typedef placement;

struct placement_node {
    struct placement_node *next;
    struct placement *placed;
} typedef placement_node;

struct frame {
    struct placement_node *head;
} typedef frame;

void color(int rgb[]) {
    float r = (1/255.0)*rgb[1];
    float g = (1/255.0)*rgb[2];
    float b = (1/255.0)*rgb[3];

    glColor3f(r, g, b);
}

void display_rect(int x, int y, int width, int height, int rgb[]) {
    color(rgb);

    glBegin(GL_POLYGON);
    glVertex2f(x, y);
    glVertex2f(x, y+height);
    glVertex2f(x+width, y+height);
    glVertex2f(x+width, y);
    glEnd();
}

void display_triangle(int x, int y, int length, int rgb[]) {
    color(rgb);

    glBegin(GL_TRIANGLES);
    glVertex2f(x, y);
    glVertex2f(x + length, y);
    glVertex2f(x + length/2, y + length/2);
    glEnd();
}

void display_ellipse(int x, int y, int width, int height, int
    rgb[]) {
    color(rgb);

    const float Pi2=2*3.141593;
    int centerX = x + width/2;
    int centerY = y + height/2;
    int i;

    glBegin(GL_TRIANGLE_FAN);
    glVertex2f(centerX, centerY);
    for(i=0; i <= 25; i++) {
        glVertex2f(
            centerX + ((width/2) * cos(i * Pi2/ 25)),
            centerY + ((height/2) * sin(i * Pi2/ 25))
        );
    }
    glEnd();
}

void display_image(int x, int y, int width, int height, char
    *filename) {
    glColor3f(1.0, 1.0, 1.0);
    glEnable (GL_BLEND);
    glBlendFunc (GL_ONE, GL_ONE_MINUS_SRC_ALPHA);
    glEnable(GL_TEXTURE_2D);
    int img_width, img_height;

    char absolutepath[1000];
    if(realpath(filename,absolutepath) == NULL) {
        printf("Invalid image: %s\n", filename);
        return;
    }
    unsigned char* image =
    SOIL_load_image(absolutepath, &img_width, &img_height, 0,
       SOIL_LOAD_RGBA);

    if(image == NULL) {
       printf("Invalid image: %s\n", filename);
       return;
    }

    GLuint gltext;
    glGenTextures(1,&gltext);
    glBindTexture(GL_TEXTURE_2D, gltext);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, img_width, img_height,
       0, GL_RGBA,
               GL_UNSIGNED_BYTE, image);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
       GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
       GL_LINEAR);

    glBegin(GL_QUADS);
    glTexCoord2i(0, 0); glVertex2i(x, y+height);
    glTexCoord2i(0, 1); glVertex2i(x, y);
    glTexCoord2i(1, 1); glVertex2i(x+width, y);
    glTexCoord2i(1, 0); glVertex2i(x+width, y+height);

    glEnd();
    glDisable(GL_TEXTURE_2D);
    glDisable(GL_BLEND);

    SOIL_free_image_data(image);
}

int render(int numFrames, frame *frames[], int fps, int width, int
    height) {
    glfwInit();

    glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);

    GLFWwindow *window = glfwCreateWindow(width, height, "PixCzar",
        NULL, NULL);

    int screenWidth, screenHeight;
    glfwGetFramebufferSize( window, &screenWidth, &screenHeight);

    if( window == NULL ) {
        printf("Failed to create GLFW window\n");
        glfwTerminate();
        return -1;
    }

    glfwMakeContextCurrent( window );
    glewExperimental = GL_TRUE;

    if( GLEW_OK != glewInit() ) {
        printf("Failed to initialize GLEW\n");
        glfwTerminate();
    }

    glViewport( 0, 0, screenWidth, screenHeight );
    glOrtho(0.0f,width, 0.0f, height, -1.0f, 1.0f);

    double spf = 1.0/fps;
    double lastDrawTime = glfwGetTime();

    int i;
    for(i=1; i<numFrames; i++) {
        if(glfwWindowShouldClose(window)) break;

        glClearColor( 1.0f, 1.0f, 1.0f, 1.0f );
        glClear( GL_COLOR_BUFFER_BIT );

        placement_node *node = frames[i]->head;
        while(node->placed) {
           if(node->placed->ref->type == 1) {
               display_rect(node->placed->x, node->placed->y,
                  node->placed->ref->width,
                          node->placed->ref->height,
                              node->placed->ref->rgb);
           } else if(node->placed->ref->type == 2) {
                display_triangle(node->placed->x, node->placed->y,
                    node->placed->ref->height,
                                 node->placed->ref->rgb);
            } else if(node->placed->ref->type == 3) {
                display_ellipse(node->placed->x, node->placed->y,
                    node->placed->ref->width,
                             node->placed->ref->height,
                                 node->placed->ref->rgb);
            } else if(node->placed->ref->type == 4) {
                display_image(node->placed->x, node->placed->y,
                    node->placed->ref->width,
                              node->placed->ref->height,
                                  node->placed->ref->filename);
            }

            node = node->next;
        }
        glfwSwapBuffers( window );

        lastDrawTime = glfwGetTime();
        glfwPollEvents();

        while(glfwGetTime() - lastDrawTime < spf);
    }

    glfwTerminate();
    return 0;
}
```

8.8. Makefile (only for main.c)

Author: Frank

```
CC = gcc -std=c99
CFLAGS = -c -g -Wall -Wextra

# LINKER_FLAGS specifies the libraries we're linking against
LINKER_FLAGS = -lGL -lglut -lGLEW -lglfw
LINKER_FLAGS = -framework GLUT -framework OpenGL -framework
    CoreFoundation -lGLEW -lglfw -lsoil
```

```
# COMPILER_FLAGS specifies the additional compilation options
    we're using
# -Wall will turn on all standard warnings
COMPILER_FLAGS = -Wall

all: main.o

main.o: main.c
   $(CC) -c -o $@ main.c $(COMPILER_FLAGS)

clean:
   rm main.o
```

8.9. compile.sh

Author: Bryan

```
#!/bin/bash
export PATH=$PATH:/usr/local/opt/llvm/bin

# needed for compilation
CC="clang"
os=`uname`
if [[ "$os" == "Darwin" ]]
then
  LIBS="-framework GLUT -framework OpenGL -framework CoreFoundation
      -lGLEW -lglfw -lsoil"
elif [[ "$os" == "Linux" ]]
then
  LIBS="-lGL -lglut -lGLEW -lglfw -lm -L./openGL -lSOIL"
else
  LIBS="-lGL -lglut -lGLEW -lglfw -lm -L./openGL -lSOIL"
  echo '`uname` not Darwin or Linux, compilation may fail'
fi
source ./include.sh

# file is command line argument, filename is file without extension
file=$1
filename=$(echo "$file" | cut -f 1 -d ".")
```

```
func=$2

create() { # generate code, need to pass in file since it could be
    modified
    ./pixczar.native "$1" > "$filename".ll
    llc "$2".ll
    eval "$CC $LIBS -o $filename.exe $filename.s opengl/main.o"
    echo "$filename.exe created"
}

# SCRIPT BEGINS HERE
if [[ $# -eq 0 ]]
then
    echo ’usage: ./compile.sh <file.pxr> [func]’
    exit 0
fi

if [ ! -e $file ]
then
    echo "file $file not found"
    exit 0
fi

if [[ "$func" == "clean" ]]
then
    echo "cleaning: $filename.s $filename.ll $filename.exe"
    rm "$filename".s "$filename".ll "$filename".exe

    include_file="$filename"_included.pxr
    if [ -e $include_file ]
    then
        echo "cleaning: $filename"_included.pxr
        rm $include_file
    fi
    exit 0
fi

echo "${reset}compiling: $file"

## following code is for handling #include
```

```
generate_includes "$file"
ret_code=$?
if [ $ret_code -eq 1 ]
then file="$(echo $file | cut -f 1 -d ".")$suffix" # use
    _included.pxr instead of original
fi
#######################################

create "$file" "$filename"

if [[ "$func" == "run" ]]
then
echo "running: $filename.exe"
./"$filename".exe
fi
```

8.10. testall.sh

Authors: Mat, Gay, Bryan, Frank

```
#!/bin/bash
export PATH=$PATH:/usr/local/opt/llvm/bin

passing_tests="tests/passing_tests/*.pxr"
failing_tests="tests/failing_tests/*.pxr"
success="SUCCESS"
failure="FAILURE"
red=‘tput setaf 1‘
green=‘tput setaf 2‘
reset=‘tput sgr0‘
suffix="_included.pxr"

CC="clang"

os=‘uname‘
if [[ "$os" == "Darwin" ]]
then
  LIBS="-framework GLUT -framework OpenGL -framework CoreFoundation
      -lGLEW -lglfw -lsoil"
elif [[ "$os" == "Linux" ]]
```

```
then
  LIBS="-lGL -lglut -lGLEW -lglfw -lm -L./openGL -lSOIL"
else
  LIBS="-lGL -lglut -lGLEW -lglfw -lm -L./openGL -lSOIL"
  echo ’‘uname‘ not Darwin or Linux, compilation may fail’
fi

./make.sh
source ./include.sh

create() { # generate code
    ./pixczar.native "$1" > "$filename".ll
    llc "$filename".ll
    eval "$CC $LIBS -o $filename.exe $filename.s opengl/main.o"
    rm "$filename".s "$filename".ll
}

for file in $passing_tests
do
  echo "${reset}running: $file"

  ## following code is for handling #include
  filename=$(echo "$file" | cut -f 1 -d ".") # for files .out .ll
      .s, etc.
  generate_includes "$file"
  ret_code=$?
  if [ $ret_code -eq 1 ]
  then file="$(echo $file | cut -f 1 -d ".")$suffix" # use
      _included.pxr instead of original
  fi
  #######################################

  outfile="$filename".out
  create "$file"
  diff "$outfile" <(./"$filename".exe)
  if  [ $? -ne 0 ];
  then
    echo ${red}$failure ${reset}
  else
    echo ${green}$success "${reset}"
  fi

  rm "$filename".exe
  if [ $ret_code -eq 1 ]
    then rm "$file"
  fi
  echo "---------------------------------------"
done

for file in $failing_tests
do
  echo "${reset}running: $file"

  ## following code is for handling #include
  filename=$(echo "$file" | cut -f 1 -d ".") # for files .out .ll
      .s, etc.
  generate_includes "$file"
  ret_code=$?
  ######################################

  outfile=$(echo "$file" | cut -f 1 -d ".")".out"
  # echo (./pixczar.native "$file" 2>&1) >&2
  diff "$outfile" <(./pixczar.native "$file" 2>&1)
  if  [ $? -eq 0 ];
  then
    echo ${green}$failure ${reset}
  else
    echo ${red}$success or NOT FOUND"${reset}"
  fi
  echo "---------------------------------------"
done
```

8.11. include.sh

Author: Bryan

```
#!/bin/bash
export PATH=$PATH:/usr/local/opt/llvm/bin

# usage: include.sh <source_file.pxr>
# output: included_source_file.pxr (generates new file with
    included_ prefix)

include="#include *.*"
quotes="\".*\""
# orig_file_name=$1

function generate_includes {
    ret_code=0
    input_filepath=$1
    name=$(echo $input_filepath | cut -f 1 -d ".")
    suffix="_included.pxr"
    local_path=$(dirname "${input_filepath}")

    while read line; do

        if [[ $line =~ $include ]]
        then
            arr=($line)

            to_include_filename=${arr[1]}

            # strip first and last quotes from filename, append
                local_path
            to_include_filename="${to_include_filename%\"}"
            to_include_filename="${to_include_filename#\"}"
            to_include_filename="$local_path/$to_include_filename"

            if [ ! -e $to_include_filename ]
            then
                echo "INCLUDE ERROR: file $to_include_filename not
                    found" >&2
            else
                to_include="$(cat $to_include_filename)"
                # echo "$line" >&2
                # echo "$to_include" >&2

                # sed ’s/"$line"/"$to_include"/’ $input_filepath >
                    "$name$suffix"
                source=$(cat $input_filepath)
                # echo "$to_include" >&2
                # echo -e hi >&2

                output="${source//"$line"/$to_include}"
                echo "$output" > "$name$suffix"
                ret_code=1
            fi
        fi
    done < $1

    if [ $ret_code -eq 0 ]
    then return 0
    else return 1
    fi
}

# generate_includes $1
```

8.12. Tests

8.12.1. Passing Tests

Authors: Frank, Bryan, Gary, Mat

```
==> tests/passing_tests/addsubtract.out <==
7
1
6.3
0.1

==> tests/passing_tests/addsubtract.pxr <==
Void main() {
   Int x = 4;
   Int y = 3;
   print(x+y);
   print(x-y);

   Float a = 3.2;
   Float b = 3.1;
   print(a+b);
   print(a-b);
}

==> tests/passing_tests/aob.out <==
Throwing Runtime Error: Out of Bound Array Access

==> tests/passing_tests/aob.pxr <==
Void main() {
  Int[] y= new Int[4];
  Int x = 4;
  y[x];
}


==> tests/passing_tests/arrayassign.out <==
100

==> tests/passing_tests/arrayassign.pxr <==
Void main() {
  Int[] arr = new Int[2];
  arr[0] = 100;
  print(arr[0]);
}

==> tests/passing_tests/arrayinit.out <==
1
2
3

==> tests/passing_tests/arrayinit.pxr <==
Void main() {
  Int[] arr = [1, 2, 3];
  print(arr[0]);
  print(arr[1]);
  print(arr[2]);
}

==> tests/passing_tests/arraylength.out <==
10
3
2
```

```
==> tests/passing_tests/arraylength.pxr <==
Void main() {
  Int[] arr = new Int[10];
  print(length(arr));
  print(length(new Int[3]));
  print(length([1,1]));
}

==> tests/passing_tests/assignmentop.out <==
3
4

==> tests/passing_tests/assignmentop.pxr <==
Void main() {
   Int x = 3;
   print(x);
   x = 4;
   print(x);
}

==> tests/passing_tests/bracketprecedence.out <==
10

==> tests/passing_tests/bracketprecedence.pxr <==
Void main() {
   Int[] arr = new Int[10];
   arr[5] = 10;
   print(arr[(2+3)]);
}

==> tests/passing_tests/break.out <==
1

==> tests/passing_tests/break.pxr <==
Void main() {
  Int x = 1;
  while(x > 0) {
    print(x);
    break;
  }
}

==> tests/passing_tests/cast.out <==
1
-1
3
-3

==> tests/passing_tests/cast.pxr <==
Void main() {
  Int x = 1;
  Int y = -1;
  Float z = 3.4;
  Float w = -3.4;
  print(~x);
  print(~y);
  print(~z);
  print(~w);
}


==> tests/passing_tests/comments.out <==
0

==> tests/passing_tests/comments.pxr <==
Void main() {
    Int x = 0;
    //x = x + 1;
    print(x);
    /*should print 0*/
}

==> tests/passing_tests/conditionals.out <==
1
2
3

==> tests/passing_tests/conditionals.pxr <==
Void main() {
    Int x = 1;
    if (x == 1){
        print(1);
    }
    else if (x == 2){
        print(2);
    }
    else{
        print(3);
    }

    x = 2;
    if (x == 1){
        print(1);
    }
    else if (x == 2){
        print(2);
    }
    else{
        print(3);
    }

    x = 3;
    if (x == 1){
        print(1);
    }
    else if (x == 2){
        print(2);
    }
    else{
        print(3);
    }
}

==> tests/passing_tests/continue.out <==
9
7
5
3
1
```

```
==> tests/passing_tests/continue.pxr <==
Void main() {
  Int x = 10;
  while(--x > 0) {
    if(x % 2 == 0) {
        continue;
    }
    print(x);
  }
}

==> tests/passing_tests/defaultvalues.out <==
0
0
0


==> tests/passing_tests/defaultvalues.pxr <==
Int x;
Float y;
String z;
Boolean b;

Void main() {
  print(x);
  print(y);
  print(b);
  print(z);
}

==> tests/passing_tests/elseif.out <==
Else if

==> tests/passing_tests/elseif.pxr <==
Void main() {
  Int check = 3;

  if(check == 2) {
    print("If");
  } else if(check == 3) {
    print("Else if");
  } else {
    print("Else");
  }
}

==> tests/passing_tests/for.out <==
Loop
Loop
Loop
Loop
Loop

==> tests/passing_tests/for.pxr <==
Void main() {
  Int x;
  for(x = 0; x < 5; x++) {
    print("Loop");
  }
}

==> tests/passing_tests/functionglobals.out <==
1

==> tests/passing_tests/functionglobals.pxr <==
Int x = 3;

Void test() {
  print(x);
}

Void main() {
  x = 1;
  test();
}

==> tests/passing_tests/globalassign.out <==
1
1.1
```

```
1
test

==> tests/passing_tests/globalassign.pxr <==
Int x = 1;
Float y = 1.1;
Boolean z = true;
String w = "test";

Void main() {
  print(x);
  print(y);
  print(z);
  print(w);
}

==> tests/passing_tests/helloworld.out <==
Hello World

==> tests/passing_tests/helloworld.pxr <==
Void main() {
   String x = "Hello World";
   print(x);
}

==> tests/passing_tests/if_types.out <==
true expression
Int
Float
String
Pix

==> tests/passing_tests/if_types.pxr <==
Void main() {
  Boolean x, y;
  x = true;
  y = true;
  if(x == y) {
    print("true expression");
  }
  if(x != y) {
    print("false expression");
  }
  if(1){
    print("Int");
  }
  if(1.0){
    print("Float");
  }
  if("s"){
    print("String");
  }
  if(new Pix()){
    print("Pix");
  }
  if(null){
    print("null");
  }
}

==> tests/passing_tests/ifelse.out <==
else cond

==> tests/passing_tests/ifelse.pxr <==
Void main() {
  String cond = "else";
  if(cond == "if") {
    print("if cond");
  } else {
    print("else cond");
  }
}

==> tests/passing_tests/include_test.out <==
3

==> tests/passing_tests/include_test.pxr <==
#include "return.pxr"

==> tests/passing_tests/inequalities.out <==
1
1
1
0
0
1
1
0
0
1

==> tests/passing_tests/inequalities.pxr <==
Void main() {
   print(3 < 4);
   print(3 <= 4);
   print(3 <= 3);
   print(3 > 4);
   print(3 >= 4);
   print(3 >= 3);
   print(3 == 3);
   print(3 == 4);
   print(3 != 3);
   print(3 != 4);

}

==> tests/passing_tests/iterate_array.out <==
1
2
3
4
5

==> tests/passing_tests/iterate_array.pxr <==
Void main() {
  Int[] x = [1,2,3,4,5];
  Int i;
  for(i = 0; i < length(x); i++) {
    print(x[i]);
  }
}

==> tests/passing_tests/logicgates.out <==
1
0
0
1
1
0
0
1

==> tests/passing_tests/logicgates.pxr <==
Void main() {
   print(true && true);
   print(true && false);
   print(false && false);
   print(true || true);
   print(true || false);
   print(false || false);
   print(true && true && false);
   print(false || false || true);
}

==> tests/passing_tests/modulo.out <==
1
0
3

==> tests/passing_tests/modulo.pxr <==
Void main() {
   Int x = 6;
   Int y = 5;
   Int z = 3;
   print(x%y);
   print(x%z);
   print(z%x);
}

==> tests/passing_tests/multdivide.out <==
18
2
2.88
2

==> tests/passing_tests/multdivide.pxr <==
Void main() {
   Int x = 6;
   Int y = 3;
   print(x*y);
   print(x/y);

   Float a = 2.4;
   Float b = 1.2;
   print(a*b);
   print(a/b);
}

==> tests/passing_tests/newline.out <==
hello

world

==> tests/passing_tests/newline.pxr <==
Void main() {
   print("hello");
   print("");
   print("world");
}

==> tests/passing_tests/operatorprecedence.out <==
15

==> tests/passing_tests/operatorprecedence.pxr <==
Void main() {
   Int x = ((1+2)+3*4);
   print(x);
}

==> tests/passing_tests/postops.out <==
1
0

==> tests/passing_tests/postops.pxr <==
Void main() {
  Int i = 0;
  print(i++);
  print(i--);
}

==> tests/passing_tests/reassign.out <==
hello
world

==> tests/passing_tests/reassign.pxr <==
Void main() {
  String x = "hello";
  print(x);
  x = "world";
  print(x);
}

==> tests/passing_tests/return.out <==
3

==> tests/passing_tests/return.pxr <==
Int retint(){
   return 3;
}


Void main() {
   print(retint());
}

==> tests/passing_tests/scope_global.out <==
30

==> tests/passing_tests/scope_global.pxr <==
Int x;
```

```
Void add10() {
    x = x + 10;
}

Void add20() {
    x = x + 20;
}

Void main() {
    x = 0;
    add10();
    add20();
    print(x);
}

==> tests/passing_tests/scope_local.out <==
10
20
0

==> tests/passing_tests/scope_local.pxr <==
Void add10(Int x) {
    x = x + 10;
    print(x);
}

Void add20(Int x) {
    x = x + 20;
    print(x);
}

Void main() {
    Int x = 0;
    add10(x);
    add20(x);
    print(x);
}
```

```
==> tests/passing_tests/unop_int.out <==
0
1
1
0

==> tests/passing_tests/unop_int.pxr <==
Void main() {
  Int x = 0;
  print(++x);
  print(x);
  print(--x);
  print(x);
}

==> tests/passing_tests/unop_v2.out <==
0
1
-3
3

==> tests/passing_tests/unop_v2.pxr <==
Void main() {
   print(!true);
   print(!false);
   print(-3);
   print(-(-3));
}

==> tests/passing_tests/while.out <==
Loop
Loop
Loop
Loop
Loop

==> tests/passing_tests/while.pxr <==
Void main() {
  Int x = 0;
  while(x < 5) {
    print("Loop");
    x = x + 1;
  }
}
==> tests/passing_tests/cast.out <==
1
-1
3
-3

==> tests/passing_tests/cast.pxr <==
Void main() {
  Int x = 1;
  Int y = -1;
  Float z = 3.4;
  Float w = -3.4;
  print(~x);
  print(~y);
  print(~z);
  print(~w);
}
```

8.12.2. Failing Tests

Authors: Frank, Bryan, Gary, Mat

```
==> tests/failing_tests/arrayassigntypecheck.out <==
Fatal error: exception Failure("illegal assignment Int = String in
    arr[0] = "WrongType"")

==> tests/failing_tests/arrayassigntypecheck.pxr <==
Void main() {
  Int[] arr = new Int[1];
  arr[0] = "WrongType";
}

==> tests/failing_tests/arrayoutofbounds.out <==
Fatal error: exception Failure("Illegal index")

==> tests/failing_tests/arrayoutofbounds.pxr <==
Void main() {
  Int[] arr = new Int[1];
  print(arr[2]);
  print(arr[-1]);
}

==> tests/failing_tests/assign_undeclared.out <==
Fatal error: exception Failure("undeclared identifier y")

==> tests/failing_tests/assign_undeclared.pxr <==
Void main() {
   Int x = y;
   print(x);
}

==> tests/failing_tests/assignfloat.out <==
Fatal error: exception Failure("illegal assignment Float = Int in
    f = 3")

==> tests/failing_tests/assignfloat.pxr <==
Void main() {
   Float f = 2.0;
   f = 3;
}

==> tests/failing_tests/assignstr.out <==
Fatal error: exception Failure("illegal assignment String = Int in
    s = 3")

==> tests/failing_tests/assignstr.pxr <==
Void main() {
   String s = "hello";
   s = 3;
}

==> tests/failing_tests/continue.out <==
Fatal error: exception Failure("Continue statement not in loop")

==> tests/failing_tests/continue.pxr <==
Void main() {
continue;
```

```
}

==> tests/failing_tests/dupefunction.out <==
Fatal error: exception Failure("duplicate function func")

==> tests/failing_tests/dupefunction.pxr <==
Void func(){
   print("i am the original");
}

Void func(){
   print("i am a duplicate");
}

Void main() {
   func();
}

==> tests/failing_tests/duplicatevar.out <==
Fatal error: exception Failure("Duplicate variable declaration x")

==> tests/failing_tests/duplicatevar.pxr <==
Void main() {
   Int x;
   Int x;
}

==> tests/failing_tests/dupvars.out <==
Fatal error: exception Failure("Duplicate variable declaration x")

==> tests/failing_tests/dupvars.pxr <==
Void main() {
Int x = 0;
Int x = 1;
}

==> tests/failing_tests/dupvarsglobal.out <==
Fatal error: exception Failure("Duplicate variable declaration x")

==> tests/failing_tests/dupvarsglobal.pxr <==
Int x = 3;

Void main() {
  Int x = 1;
  print(x);
}


==> tests/failing_tests/emptyrgb.out <==
Fatal error: exception Parsing.Parse_error

==> tests/failing_tests/emptyrgb.pxr <==
Void main() {
  Pix test = new Pix();
  test.makeTriangle(200, [])
}
==> tests/failing_tests/float_to_string.out <==
Fatal error: exception Failure("illegal assignment Float = String
    in x = "hello"")

==> tests/failing_tests/float_to_string.pxr <==
Void main() {
   Float x = 3.5;
   x = "hello";
   print(x);
}

==> tests/failing_tests/frameoutofbounds.out <==
Fatal error: exception Failure("Illegal index")

==> tests/failing_tests/frameoutofbounds.pxr <==
Void main() {
   Frame[] frames = new Frame[1];
   frames[0] = new Frame();
   frames[1] = new Frame();
}
==> tests/failing_tests/functioncreation.out <==
Fatal error: exception Parsing.Parse_error

==> tests/failing_tests/functioncreation.pxr <==
Int test1{}


Int test3(hello,goodbye){}



==> tests/failing_tests/functioncreation2.out <==
Fatal error: exception Parsing.Parse_error

==> tests/failing_tests/functioncreation2.pxr <==
Int test2()

==> tests/failing_tests/functioncreation3.out <==
Fatal error: exception Parsing.Parse_error

==> tests/failing_tests/functioncreation3.pxr <==
Int test3(hello,goodbye){}


==> tests/failing_tests/global_invalid.out <==
Fatal error: exception Failure("Invalid type for global variable")

==> tests/failing_tests/global_invalid.pxr <==
Pix p;

Void main() {
}

==> tests/failing_tests/include_not_found.out <==
Fatal error: exception Parsing.Parse_error

==> tests/failing_tests/include_not_found.pxr <==
#include "missing.pxr"
Void main() {}

==> tests/failing_tests/int_to_float.out <==
Fatal error: exception Failure("illegal assignment Int = Float in
   x = 3.5")
```

```
==> tests/failing_tests/int_to_float.pxr <==
Void main() {
   Int x = 3;
   x = 3.5;
   print(x);
}

==> tests/failing_tests/int_to_string.out <==
Fatal error: exception Failure("illegal assignment Int = String in
    x = "hello"")

==> tests/failing_tests/int_to_string.pxr <==
Void main() {
   Int x = 3;
   x = "hello";
   print(x);
}

==> tests/failing_tests/intarr_to_int.out <==
Fatal error: exception Failure("LHS type of Int[] not the same as
    RHS type of Int")

==> tests/failing_tests/intarr_to_int.pxr <==
Void main() {
   Int[] arr = 8;
}

==> tests/failing_tests/invalid_placement_pos.out <==
Fatal error: exception Failure("illegal argument found Int
    expected Pix in x")

==> tests/failing_tests/invalid_placement_pos.pxr <==
Void main() {
   Pix tri = new Pix();
   tri.makeTriangle(200,[200,200,200]);
   Int x = 3;
   Placement placed1 = new Placement(tri,200,300);
   Placement placed2 = new Placement(x,300,300);
}
```

```
==> tests/failing_tests/invalidaddplacement.out <==
Fatal error: exception Failure("illegal object function call
    frames[0].addPlacement(test);
")

==> tests/failing_tests/invalidaddplacement.pxr <==
Void main() {
   Pix test = new Pix();
   Frame[] frames = new Frame[1];
   frames[0].addPlacement(test);
   //add pix instead of placement
}
==> tests/failing_tests/invalidellipse.out <==
Fatal error: exception Failure("illegal object function call
    el.makeEllipse(200, [200,200,200]);
")

==> tests/failing_tests/invalidellipse.pxr <==
Void main() {
  Pix el = new Pix();
  el.makeEllipse(200,[200,200,200]);
}
==> tests/failing_tests/invalidplacement.out <==
Fatal error: exception Failure("undeclared identifier text")

==> tests/failing_tests/invalidplacement.pxr <==
Void main() {
  Pix test = new Pix();
  Placement placed = new Placement(text,200,"yo");
}

==> tests/failing_tests/invalidrect.out <==
Fatal error: exception Failure("illegal object function call
    rect.makeRectangle(200, [200,200,200]);
")

==> tests/failing_tests/invalidrect.pxr <==
Void main() {
  Pix rect = new Pix();
  rect.makeRectangle(200,[200,200,200]);
}
==> tests/failing_tests/invalidrender.out <==
Fatal error: exception Failure("expecting 4 arguments in
    render(frames, 1, -(1))")

==> tests/failing_tests/invalidrender.pxr <==
Void main() {
   Pix tri = new Pix();
   tri.makeTriangle(200,[200,200,200]);
   Pix rect = new Pix();
   rect.makeRectangle(100,200,[200,200,200]);
   Placement placed = new Placement(tri,100,300);
   Placement placed1 = new Placement(rect,200,300);
   Placement placed2 = new Placement(tri,300,300);
   Frame[] frames = new Frame[2];
   frames[0] = new Frame();
   frames[1] = new Frame();
   frames[2] = new Frame();
   frames[0].addPlacement(placed);
   frames[1].addPlacement(placed1);
   frames[2].addPlacement(placed2);
   print(render(frames,1, -1));
   //-1 fps
}

==> tests/failing_tests/invalidrgb.out <==
Fatal error: exception Failure("Invalid rgb input")

==> tests/failing_tests/invalidrgb.pxr <==
Void main() {
  Int[] arr = [200,200,200,200];
  Pix test = new Pix();
  test.makeTriangle(200,arr);

}
==> tests/failing_tests/invalidtriangle.out <==
Fatal error: exception Failure("illegal object function call
    tri.makeTriangle(200, 200, [200,200,200]);
")
```

```
==> tests/failing_tests/invalidtriangle.pxr <==
Void main() {
  Pix tri = new Pix();
  tri.makeTriangle(200,200,[200,200,200]);
}
==> tests/failing_tests/missingmain.out <==
Fatal error: exception Failure("unrecognized function main")

==> tests/failing_tests/missingmain.pxr <==
Void notmain(){
   print("hello");
}
==> tests/failing_tests/negativefps.out <==
Fatal error: exception Failure("Illegal index")

==> tests/failing_tests/negativefps.pxr <==
Void main() {
   Pix tri = new Pix();
   tri.makeTriangle(200,[200,200,200]);
   Pix rect = new Pix();
   rect.makeRectangle(100,200,[200,200,200]);
   Placement placed = new Placement(tri,100,300);
   Placement placed1 = new Placement(rect,200,300);
   Placement placed2 = new Placement(tri,300,300);
   Frame[] frames = new Frame[2];
   frames[0] = new Frame();
   frames[1] = new Frame();
   frames[2] = new Frame();
   frames[0].addPlacement(placed);
   frames[1].addPlacement(placed1);
   frames[2].addPlacement(placed2);
   print(render(frames,-1, 800, 600));
   //-1 fps
}

==> tests/failing_tests/negativeframesize.out <==
Fatal error: exception Failure("Illegal index")

==> tests/failing_tests/negativeframesize.pxr <==
Void main() {
   Pix tri = new Pix();
   tri.makeTriangle(200,[200,200,200]);
   Pix rect = new Pix();
   rect.makeRectangle(100,200,[200,200,200]);
   Placement placed = new Placement(tri,100,300);
   Placement placed1 = new Placement(rect,200,300);
   Placement placed2 = new Placement(tri,300,300);
   Frame[] frames = new Frame[2];
   frames[0] = new Frame();
   frames[1] = new Frame();
   frames[2] = new Frame();
   frames[0].addPlacement(placed);
   frames[1].addPlacement(placed1);
   frames[2].addPlacement(placed2);
   print(render(frames,1, -1, -1));
   //-1 fps
}

==> tests/failing_tests/negativelength.out <==
Fatal error: exception Parsing.Parse_error

==> tests/failing_tests/negativelength.pxr <==
Void main() {
  Pix test = new Pix();
  test.makeTriangle(-1,[200,200,200])
}
==> tests/failing_tests/newcallargs.out <==
Fatal error: exception Failure("expecting 0 arguments in new
    Pix(x)")

==> tests/failing_tests/newcallargs.pxr <==
Void main() {
  Int x = 0;
  Pix pix = new Pix(x);
}

==> tests/failing_tests/outofscope.out <==
Fatal error: exception Failure("undeclared identifier x")

==> tests/failing_tests/outofscope.pxr <==
Void Hello(){
   print(x);
}

Void main() {
   Int x = 3;
   Hello();
}

==> tests/failing_tests/redefine_print.out <==
Fatal error: exception Failure("expecting 1 argument in print")

==> tests/failing_tests/redefine_print.pxr <==
Int x = 0;
Void print(){
   x = 1;
}

Void main() {
   print();
}

==> tests/failing_tests/returntype.out <==
Fatal error: exception Failure("return gives Int expected Void in
    x")

==> tests/failing_tests/returntype.pxr <==
Void main() {
   Int x = 0;
   return x;
}

==> tests/failing_tests/typecomp.out <==
Fatal error: exception Failure("illegal binary operator Int >
    String in x > y")

==> tests/failing_tests/typecomp.pxr <==
Void main() {
   Int x = 3;
   String y = "three";
   print(x > y);
}

==> tests/failing_tests/undeclaredvar.out <==
Fatal error: exception Failure("undeclared identifier y")

==> tests/failing_tests/undeclaredvar.pxr <==
Void main() {
   Int x = 0;
   print(y);
}

==> tests/failing_tests/wrongforloop.out <==
Fatal error: exception Parsing.Parse_error

==> tests/failing_tests/wrongforloop.pxr <==
Void main() {
   for(Int i=0; i<"hello"; i++){
      print("hi");
   }

   for (Int i="zero"; i<"hello"; i++){
      print("hi");
   }

}
```

8.13. stdlib.pxr

Author: Frank

```
Int WIDTH = 800;
Int HEIGHT = 800;

Void fillFrames(Frame[] frames, Placement placement, Int start,
    Int end) {
  Int i;
  for(i = start; i < end; i++) {
      if(i < length(frames)) {
        frames[i].addPlacement(placement);
```

```
     }
  }
}

Void keyFrame(Frame[] frames, Int start, Pix obj, Int[] from,
    Int[] to, Int duration) {
    Int xTimeStep = (to[0]-from[0])/duration;
    Int yTimeStep = (to[1]-from[1])/duration;

    Int i;
    for(i = 0; i < duration; i++) {
       Placement plcmt = new Placement(obj, from[0] + i*xTimeStep,
              from[1] + i*yTimeStep);
       if(i+start < length(frames)) {
           frames[i+start].addPlacement(plcmt);
       }
    }
}

Int accelerationDistance(Int t, Int a) {
  Int half = a/2;
  return half * t * t;
}

Int[] projectile(Frame[] frames, Int start, Pix obj, Int xSpeed,
    Int width,
       Int height, Int endHeight, Int gravity) {
    //g in pixel/frame^2
    Int d = accelerationDistance(1, gravity);

   Int i;
   for(i=0; (height > endHeight && gravity > 0) || (height <
      endHeight && gravity < 0); i++) {
     height = height - d;

     if(i+start < length(frames)) {
       frames[i+start].addPlacement(new Placement(obj, width +
           i*xSpeed, height));
     } else {
       break;
```

```
        }
    }

    return [i+start, width + (i-1)*xSpeed];
}
```