Seaflow Language Final Report

Ho Sanlok Lee, Rohan Arora, Junyang Jin, Sarah Seidman

{hl3436, ra3091, jj3132}@columbia.edu, ss5311@barnard.edu

1 Introduction

Seaflow is a declarative language designed to address the state management conundrum through Reactive Programming. Reactive Programming has gained huge traction over the past decade thanks to ReactiveX implementations on top of popular programming languages such as Java and JavaScript. However, the imperative nature of these source languages and lack of understanding of Reactive Programming by the application developers lead to complex implementations where multiple programming styles are mixed-used, tightly coupled within a single implementation. Seaflow is invented to resolve this problem by supporting core principles of ReactiveX natively and providing simple C-like syntax to work with data streams.

Our goals include:

1. All data types are immutable, with the exception being observables, no pointers
2. The creation of Observble data streams should be simple
3. Natively support core principles in the ReactiveX specification

2 Seaflow Tutorial

2.1 Environment Setup

We suggest you use the docker image, which can be invoked with this command:

```
docker run --rm -it -v ${PWD}:/home/seaflow -w=/home/seaflow
columbiasedwards/plt
```

Alternate instructions can be found in the readme of our git repository.

2.2 Downloading and Building Seaflow

You can clone our git repository using this command:

```
git clone https://github.com/sarahseidman/seaflow.git
```

Then cd into the directory and type `make`.

2.3 Compiling a Simple Program

Here is a simple program in Seaflow.

```
int fib(int n) {
    int y = if (x > 2) fib(x-1) + fib(x-2) else 1;
    return y;
}
printi(fib(10));
```

Save this program into a .flo file. To compile it, you can use our provided shell script.
`./seaflow fibonacci.flo`

The above line will produce an executable file called fibonacci. Run the executable like this:
`./fibonacci`

2.4 Debugging Options

The seaflow.native executable can be run with several options.

`./seaflow.native -a prog.flo`
Generates the AST.

`./seaflow.native -s prog.flo`
Generates the SAST.

`./seaflow.native -c prog.flo`
Generates LLVM code.

3 Language Manual

3.1 Lexical Conventions

3.1.1 Comments

We use the characters /* to introduce a comment, which terminates with the characters */. Any tokens between /* and */ are considered part of a comment and are not parsed.

```
/*
* This is a comment
* with multiple lines
*/

int x = 4;    /* This is also a comment */
```

3.1.2 Identifiers

Identifiers must begin with a lowercase letter and should only contain ASCII letters, digits and underscores. For observables, the identifier should be prepended with the "$" symbol.
Identifiers: `['a'-'z']['a'-'z' 'A'-'Z' '0'-'9' '_']*`
Struct identifiers: `['A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*`
Observable identifiers: `'$'['a'-'z']['a'-'z' 'A'-'Z' '0'-'9' '_']*`

3.1.3 Keywords

Seaflow keywords include:
`char, else, float, if, int, null, return, struct, void, ($), combine, subscribe, map, complete`

3.1.4 Literals

Literals represent strings or one of Seaflow's primitive types: float, char, and int.
int: `[0-9]+`
float: `([0-9]+.[0-9]+) | ([0-9]+)`
char: `['a'-'z'] | ['A'-'Z'] | ['0'-'9'] | ' ' | '!' | ['#'-'~']`
string: `"\""char*"\""`

Each literal type has a corresponding built-in print function.

```
int i = 5;
printi(i);

float f = 5.5;
printf(f);
```

```
char c = 'c';
printc(c);

char[] str = "hello world";
prints(str);
```

3.1.5 Struct

A struct is a combined representation of constituent primitives. Structs cannot have observables as members. Struct declarators must begin with a capital letter; otherwise they have the same rules as identifiers for what characters are allowed. They are defined as follows:

```
struct <declarator> {[type-list parameter-list]};
```

Once a struct is created, it is immutable.

```
struct Foo {
    int field1;
    char field2;
    char[] name;
    (int)->(int) func;
};

struct Foo f = { 1, 'c', "name", func };
```

Structs that have the same body can be substituted for each other.

```
struct A {
    int field1;
    char field2;
};
struct B {
    int field3;
    char field4;
};
```

```
struct A one = { 1, 'c'};
struct B two = one;

int func(struct A x) { return A.field1; }

func(two);
```

3.1.6 Array

An array is a collection of primitives, structs, or arrays. They cannot contain observables. Arrays are declared as follows:

```
<type>[] <identifier>;
```

Once an array is declared, it is immutable. To initialize an array with values, include a comma-separated list of expressions between square brackets. An array can also be initialized with the values of an existing array.

```
int[] arr = [ 1, 2, 3, 4, 5 ];

arr[0] = 10;    /* this is not allowed */

int[] arr2 = arr;
```

Arrays have a fixed `length` attribute.

```
int[] arr = [ 1, 2, 3, 4, 5 ];
int len = arr.length;    /* len = 5 */
```

Operations that can be performed on named arrays can also be performed on array literals.

```
int i = [ 1, 2, 3, 4, 5 ][0];
int len = [0.5, 3.4, 1.1].length;    /* len = 3 */
```

String literals are equivalent to char arrays.

```
char[] s = "hello world";
```

Arrays can be concatenated using the + operator.

```
int[] a = [1,2,3];
int[] b = [4,5,6];
int[] c = a + b;    /* len = 6 */
```

3.2 Expressions and Statements

3.2.1 Expressions

3.2.1.1 Primary Expressions

Followings are considered primary expressions and they are evaluated left to right:

<primary-expression>:
    <identifier>
    <literal>
    <primary-expression>[<expression>]   # array indexing
    <primary-expression>.<member-of-structure>
    <function-identifier>(<expression-list>) # function call

3.2.1.2 Operators

The four basic arithmetic <operator> are
    + addition
    - subtraction
    * multiplication
    / division.

These are all left-associative binary operators. Multiplication and division have a higher precedence than addition and subtraction.
The relational <operator> are

== equality
!= inequality
< less than
> greater than
<= less than or equal to
>= greater than or equal to.

These are binary operators and have lower precedence than all arithmetic operators.
The assignment operator is =, and is right associative. On the left hand side must be either an observable or the declaration of an identifier.

<expression>:
<expression> <operator> <expression>

3.2.4 Statements

3.2.4.1 Declaration

Variables are declared with the following syntax:

<declaration>:
<type-specifier> <identifier> = <expression>

```
int a = 5;
```

<identifier> must be unique to its scope, and all non-observable variables must be initialized at the time of declaration.

3.2.4.2 Immutability

All non-observables are immutable after they are created. Also, all variables once they are initialized cannot be reassigned with a different value.

```
int a = 0;
a = 4;   /* This line yields a compile-time error */
```

3.2.4.3 Observable re-assignment

Observables can be reassigned using following syntax:

<observable-identifier> = <expression>

3.2.4.4 Conditional statement

Conditional statement in Seaflow looks like this:

<conditional-expression>:
<type-specifier> <identifier> = if (<expression$_c$>) <expression$_1$> else <expression$_2$>

If <expression> is evaluated to true, the <conditional-expression> is evaluated to <expression$_1$>, and otherwise it is evaluated to <expression$_2$>

And therefore following example is possible:

```
char letterGrade = if (score > 60) 'p' else 'f';
```

3.3 Functions

3.3.1 Declaration

A Seaflow function is similar to a C function. Each function takes a list of fixed-type arguments and returns a fixed-type value. The declaration consists of two parts: a function declarator and a function body. All function names must start with a lowercase letter.

Function declarator:

<return-type> declarator ([type-list parameter-list])

Function body:

{declaration-list statement-list}

Simple example:

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

3.3.2 Scope rules

Identifiers declared outside of any function are visible throughout the program after their declaration. The scope of identifiers declared in the declaration of a block is limited to within that block. It is an error to redefine an identifier that already exists in the current scope. Observables are all visible at the global level.

3.3.3 Higher-Order Functions

Seaflow supports higher-order functions. Higher-order functions can be defined either in place or from declaration. Passed-in functions must be typed on both the upstream and downstream values.

Higher-order functions example:

```
int function(int x, (int)->(int) func) {
    return func(x);
}

function(10, (int x)->{ return x + 10; });
```

3.3.3.1 Anonymous Functions

One can declare an anonymous function by following this format:

(<input_type> <input_name>) -> {<statements>}

and pass it into higher order functions as a parameter.

Anonymous function example:

```
function(10, (int x)->{ return x + 10; });
```

The type signature of anonymous functions will be inferred at the compile time.

3.4 Observables

Observables in Seaflow are special types of objects that can internally hold another object. When the state of the observable changes, the observable sends notification to all of its “observers” that are subscribed to it. Observables can be defined only in the global scope.

3.4.1 Declaration

One can declare an observable by using a observable identifier that starts with a dollar sign, $, followed by a normal identifier. An observable must have an initial value at the declaration time. This can be expressed in following form:

<type-specifier> <observable-identifer> = <non-obsevable-expression>;

```
int $a = 0;
```

We define <observable-expression> as an expression that is evaluated to an observable, and <non-observable-expression> as an expression that is evaluated to a normal non-observable.

3.4.2 Subscription

An <observer> can receive notifications from an <observable> by subscribing to the observable. Observer can be a function with the same input type, or an observer of the same type.

<observer>:
    <function>
    <observable>

<subscription>:
    subscribe(<function>, <observable>)

```
int $a;

void observer(int num) {
    printi(num);
}

subscribe(observer, $a);
```

3.4.3 Methods

3.4.3.1 Map

An <observable> of type T can call the map function as follows:

map((T)->X func), $T obs)

Where the higher-order function passed to the map function takes an argument of the same type T, with the return value being of any type X. The map function returns a new observable with type X. The function `func` is called for each upstream value and the returned value will be passed to the downstream.

In addition, initializing an observable to some function of another observable is equivalent to calling the map function.

```
int $c = map((int x)->{ return x + 10; }, $b);

/* or equivalently */

$c = $b + 10;
```

3.4.3.2 Combine

An <observable> of type T can call the combine function as follows:

combine($T obs, $S obs, (T, S)->X func)

Where the first argument to the combine function is another observable of type S. The second argument is a function which takes two observables of types T and S, with the return value being any type X. The combine function returns a new observable with type X. The function `func` is called for each upstream value and the returned value will be passed to the downstream.

In addition, initializing an observable to some expression containing two observables is equivalent to calling the combine function.

```
int $d = combine((int x, int y)->{return x + y;}, $b, $c);

/* or equivalently */

$d = $b + $c /* type must be compatible when shorthand version is used */

$e = $f + $g * $h - $m / $k /* can chain combine calls using shorthand */
```

3.4.3.3 Complete

An <observable> can call the complete function as follows:

complete($T obs)

Any subscriptions that the observable had will be removed.

```
complete($d);
```

3.4.4 Operator overloading and Observable expressions

An <observable-expression> is an expression that contains one or more observables. An <observable-expression> is evaluated to an observable. Any other expression--all expressions that we discussed so far--is a <non-observable-expression>.

<observable-expression>
    <observable-expression> <operator> <non-observable-expression>

<non-observable-expression> <operator> <observable-expression>
<observable-expression> <operator> <observable-expression>

The base type of the observable-expression, the operator, and the non-observable-expression must be compatible.

The "=" operator with an observable on the left hand side has different behavior depending whether the right hand side of the "=" operator is an <observable-expression> or <non-observable-expression>

<observable> = <non-observable-expression>

This is an assignment statement that will set the internal object of the <observable> to a newly evaluated value of <non-observable-expression>.

<observable> = <observable-expression>

This is an assignment statement that will replace the reference of the observable identifier with the observable that is returned by the <observable-expression>.

3.5 Conversions

Relational operators cause implicit conversion between types. When floats and ints are compared, int is promoted to float. For example:

```
int i = 5;
float f = 5.5;

/* i is converted to float */
i == f
f > i
```

Implicit conversion also in arithmetic binary operations where the operands are some combination of int and float. The result of such an expression is a float.

```
int a = 5;
float b = 1005.5;

float c = a * b;
```

4 Project Plan

4.1 Development Process

4.1.1 Planning

We exchanged weekly emails with our TA, Evan Mesterhazy, to update him on the status of our project, ask any questions that arose during the week, and sometimes ask for his guidance on how best to move forward.

Our team met once or twice a week for the most of the semester, and then more frequently as we neared the deadline. We used Slack for day-to-day communication and collaborated on our LRM (and this report) via Google Docs. We also used a spreadsheet to keep track of features to be implemented and other to-do items, in which we tracked status, who was assigned to work on the item, and whether (for new features) tests had been written.

4.1.2 Specifications & Development

The goal of Seaflow was to implement reactive programming as something native to the language, and using a simple, C-like syntax. After writing the scanner and parser, we added incrementally to semant.ml and codgen.ml for each new feature we would support. The person who added a feature to semant would be the same person who added it to codegen - this way we all became intimately familiar with both files.

4.1.3 Testing

Test cases were written before each feature was implemented in order to catch errors as early as possible and aid in the development process. After a particular feature was marked as completed, all previous test cases were run and confirmed to pass before merging with the main branch. For each feature, positive cases which correspond to valid programs and negative test cases which

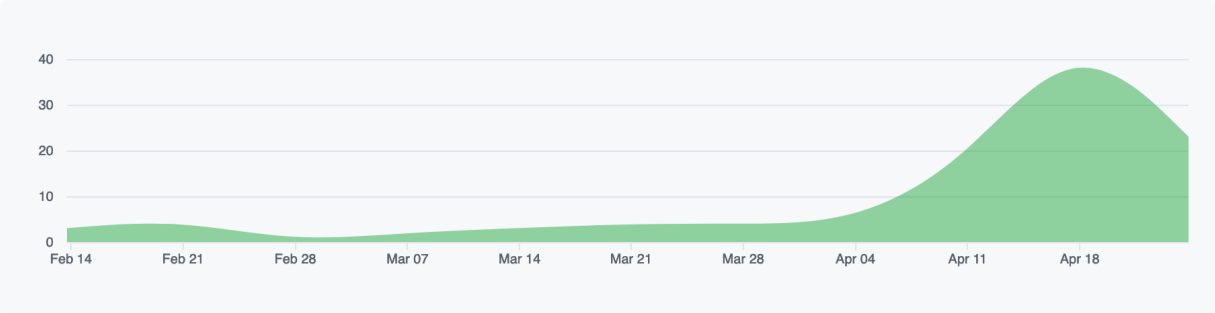
correspond to invalid programs were written. In general, rather than writing large test cases for each component, cases were split up into multiple files where possible to make it easier to pinpoint the source of errors. If the cause of a test case failure was unclear, experimentation helped to determine the conditions under which the error surfaced. Although an attempt was made to have thorough test cases from the start, there were instances where additional test cases written post-completion of a component surfaced new errors. In these cases, the errors were promptly communicated and the team prioritized their resolution before moving on to additional components.

4.2 Style

These are the guidelines we followed while developing our compiler.

- Indent using 2 spaces
- Keep lines under 100 characters
- Use descriptive identifier names
- Use underscores in identifier names instead of camel case, other than for AST and SAST types
- All SAST types are prefaced with ‘S’
- Don’t push new features without accompanying (passing) test cases

4.3 Project Timeline



This was our approximate timeline:

Date	**Milestone**
January 26	Language defined

January 31	Project Proposal
February 20	Language Reference Manual
February 24	Lexer and parser complete
March 13	AST
March 22	“Hello world” program
March 26	Global variables & basic functions
April 15	Structs & arrays
April 23	Observables

4.4 Roles & Responsibilities

Rohan Arora, Test Designer
Junyang Jin, System Architect
Sanlok Ho Lee, Language Guru
Sarah Seidman, Manager

In our design approach, all team members touched seaflowparse.mly, ast.ml, sast.ml, semant.ml, and codegen.ml as we implemented different features. The parser and scanner were implemented in meetings for which our entire group was present. Here is the breakdown otherwise:

Global variable declaration and reference	All
Binary and unary operators (including implicit conversion)	Rohan
Function declaration	All
Higher-order / anonymous functions	Sanlok
If statements	Rohan
Structs	Sarah
Arrays	Sarah
Observables	Sanlok

Observable operations: subscribe(), map(), combine()	Sanlok
Observable operation: complete()	Junyang
Pretty-printing for AST and SAST	Sarah
utils.c (array concatenation, print functions), seaflow (shell script adapted from testall.sh)	Sarah
Language Reference Manual	All
Test architecture	Rohan
Integration tests	All
Demo code	Sanlok, Sarah, Junyang
Final Report	All

4.5 Software Development Environment

Libraries and languages:
OCaml version 4.05.0
OCaml LLVM version 10.0.0
OCamlyacc version 4.05.0
OCamllex version 4.05.0
C, stdio.h libraryjj3132@

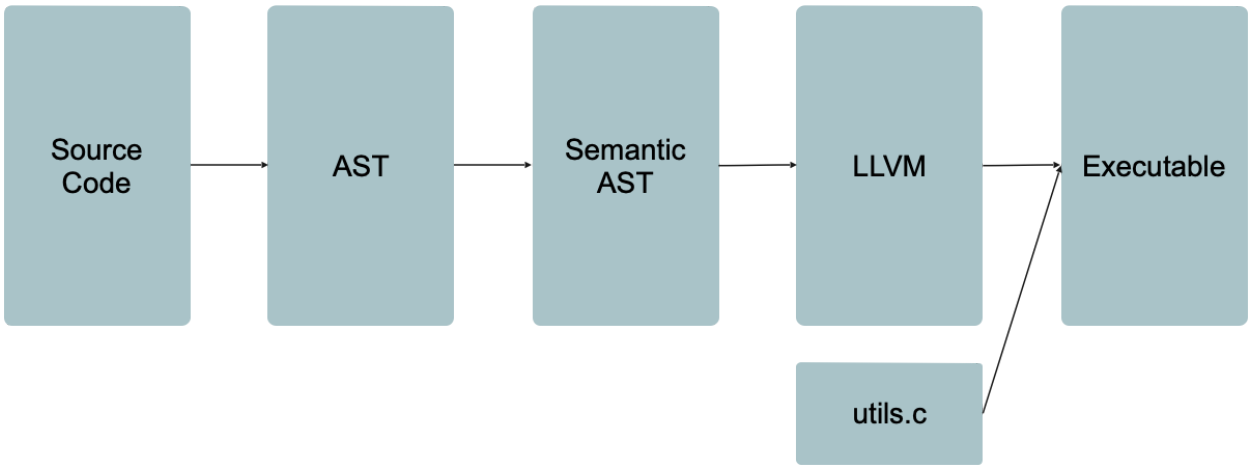
A GitHub repository was used for version control. We collaborated on text-based assignments such as this final report via Google Docs and often used VSCode LiveShare to collaborate on programming during our meetings.

4.6 Project Log

See appendix item 8.2 for our entire git log.

5 Architectural Design

5.1 Compiler

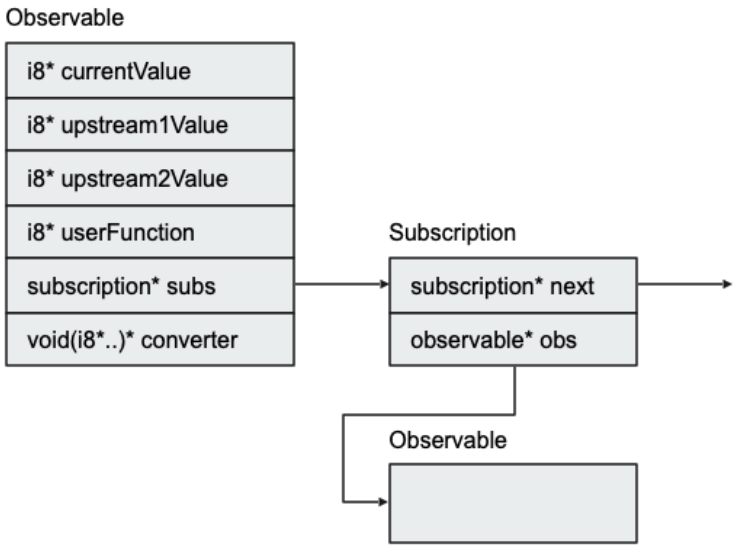


Source codes are translated into Abstract syntax trees, then translated into LLVM IR

5.2 Observables

Observables in Seaflow is inspired by the “BehaviorSubject” in the ReactiveX specification: http://reactivex.io/documentation/subject.html

5.2.1 Structure



When a Seaflow Observable is initialized, the Seaflow backend will create an Observable structure in the Heap space. Each Observable structure will contain 6 pointers:

- i8* currentValue
    - A void pointer to the current value of this observable.
- i8* upstream1Value
- i8* upstream2Value
    - Void pointers to upstream values. An Observable structure can have zero, one, or two upstream observables at a given time. Observable keeps a pointer to where the upstreams' values are stored, so that the observable can peek at them when computing the new value.
- i8* userFunction
    - A void pointer to a user defined function that is used to update the currentValue. During the propagation of the events, this function is used to compute the new "currentValue".
    - Function's return type must match with the Observable's inner type. Function's input type(s) must match with the Upstreams' inner type.
- subscription* subs
    - A pointer to the linked list of Subscription structures.
- i8* (i8* i8* i8* i8*)* converter
    - A pointer to the converter function. Observables store pointers in void types. Converter functions are responsible for casting these pointers to their original

types, loading the upstream values, calling the function, and storing the result to back to the currentValue store.

- Converters are defined during the compile time as they are needed. For example, if the Seaflow source code has an int type observable that has two char type upstreams, Seaflow will attach a converter function that converts void pointers to int*, char*, char*, and int(char*, char*)* types.

5.2.2 Graph Generation

Following Seaflow code:

```
int $b = $a + 1;
```

is internally translated into:

```
int $b = map((int x)->{ return x + 1; }, $a);
```

This code:

```
int $c = $a + $c;
```

is internally translated into:

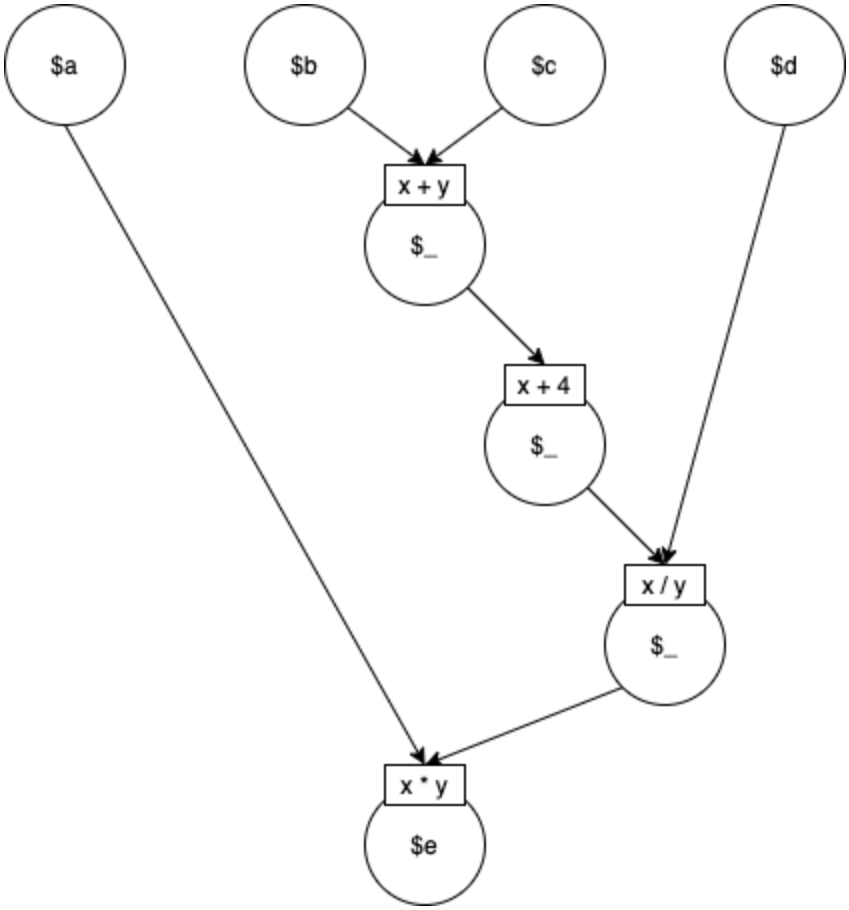
```
int $c = combine((int x, int y)->{ return x + y;}, $a, $b);
```

Both map and combine expressions are observable expressions. Observable expressions behave syntactically in a similar way as the normal expressions.

For example, for following observable statement:

```
int $e = $a * (($b + $c + 4) / $d);
```

Seaflow generates following observable graph in the backend:

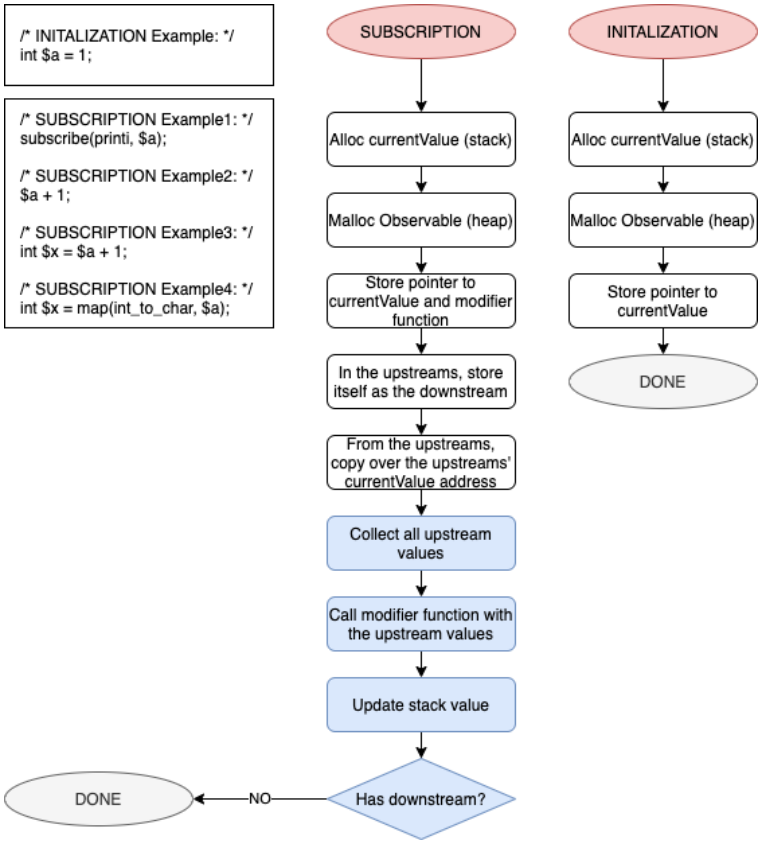


`$_` are hidden observables that are not directly reachable from the Seaflow code. This design allows Seaflow Observables to subscribe to multiple upstream observables, even though the internal representation of the Observables can only subscribe to at most two upstreams at a time. Any change in any of the upstream observables will trigger the downstream, `$e`, to be updated.

5.2.3 Initialization and Subscription

When a new Observable is initialized, a new Observable will be allocated in the heap.

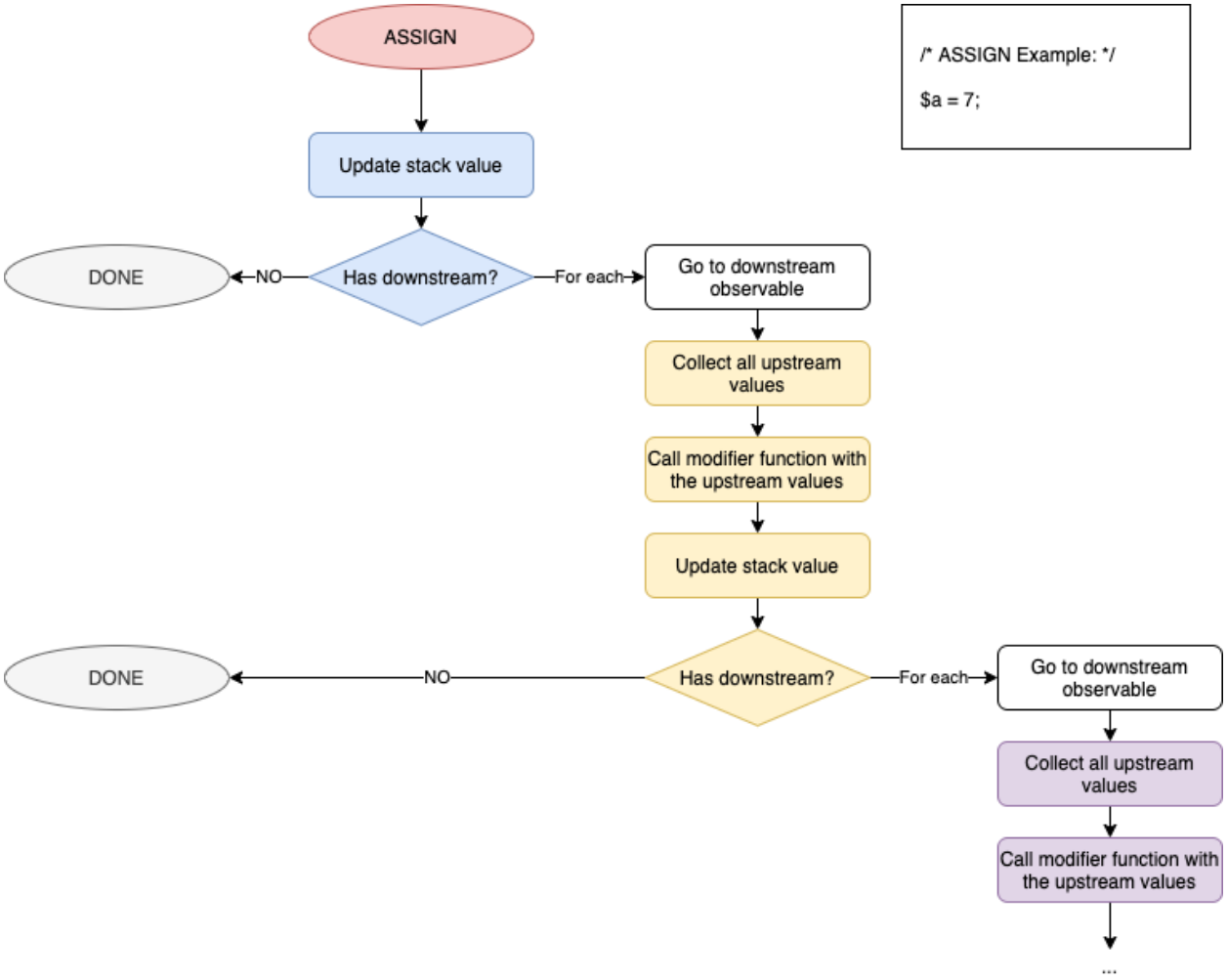
Subscription process begins when there is a “subscribe”, “map”, or “combine” call. Subscription involves creating a new observable, storing the function pointer, and lastly invoking the function for the first time.



Flow chart of Initialization and Subscription process

5.2.4 Data Propagation

When new value is assigned to an Observable, the propagation process begins. During the propagation, all downstream observables are recursively traversed in depth-first order and their values are updated.



Propagation flow chart for an ASSIGN statement

5.2.5 Observables in the Parser

Seaflow differentiates

- Observable statements vs non-observable statements
- Observable expressions vs non-observable expressions

at the Parser level. Observable statements and expressions can only appear in the global scope. This means that all statements and expressions appear in a function cannot mutate and effectively forces developers to write pure functions (except for the print call).

```
program:
 glob_lines EOF { List.rev $1 }
```

```
glob_lines:
    { [] }
 | glob_lines glob_line { $2 :: $1 }

glob_line:
 | fdecl    { Fdecl($1) }
 | stmt     { Stmt($1) }            (* non-observable statements *)
 | obs_stmt { Obs_Stmt($1) }        (* observable statements     *)
```

5.2.6 Limitations

There are several limitations in the current Observable design in Seaflow.

- Subscription can only happen at the creation of Observable

Currently Seaflow does not have any mechanism that allows developers to make connections between two existing Observables. All subscriptions must be known at the initialization stage of the Observables. While this is a limitation, it has a positive side effect of preventing cycles in the Observable graph.

- Downstream can be changed multiple times for a single upstream change:

In following example:

```
int $a = 0;
int $b = $a + 1;
int $c = $a + $b;
```

When `$a` changes, `$c` will change twice, one triggered as a child of $a and another by $b. While this might not be a critical issue of language since the eventual state of the observables is consistent. This issue can be prevented by propagating update events in topological order, and marking the Observables that are already visited.

- Single threaded

Current Seaflow design assumes that the language will run on a single thread. In order to be used in multi-threaded environments, the Observable needs to be able to buffer upstream values that runs faster than downstream, and needs locking/unlocking mechanism so that observables can be safely accessed by multiple threads.

6 Test Plan

6.1 Source to Target

Source program 1:

```
float $relative_humidity = 70.0;
float $temperature       = 20.0;
char  $temp_type         = 'c';    /* celcius */

float fahrenheit_converter (float temp, char type) {
   float adjusted = if (type == 'c') temp else (temp - 32) / 1.8;
   return adjusted;
}


float $t = combine(fahrenheit_converter, $temperature, $temp_type);


float $out = $t - (100 - $relative_humidity) / 5;

subscribe((float x)->{
   prints("New dew point:");
   printf(x);
   prints("");
   return;
}, $out);


$relative_humidity = 60.0;
$relative_humidity = 50.0;

$temperature = 70.0;
$temp_type = 'f';          /* fahrenheit */
```

Target result 1:

```
; ModuleID = 'Seaflow'
source_filename = "Seaflow"

%observable = type { i8*, i8*, i8*, i8*, %subscription*, i32, void (i8*, i8*, i8*, i8*)* }
%subscription = type { %subscription*, %observable* }

@printi = global i32 (i32)* null
@printc = global i32 (i8)* null
@printd = global i32 (double)* null
@prints = global i8** (i8**)* null
@target = global double 0.000000e+00
@"$relative_humidity" = global %observable* null
@target.1 = global double 0.000000e+00
@"$temperature" = global %observable* null
@target.2 = global i8 0
@"$temp_type" = global %observable* null
@fahrenheit_converter = global double (double, i8)* null
@target.3 = global double 0.000000e+00
@sub_obs = global %observable* null
@"$t" = global %observable* null
@target.7 = global double 0.000000e+00
@sub_obs.8 = global %observable* null
@target.10 = global double 0.000000e+00
@sub_obs.11 = global %observable* null
@target.13 = global double 0.000000e+00
@sub_obs.14 = global %observable* null
@"$out" = global %observable* null
@target.17 = global i8 0
@sub_obs.18 = global %observable* null

define i32 @main() {
entry:
```

```
  store i32 (i32)* @print_int, i32 (i32)** @printi
  store i32 (i8)* @print_char, i32 (i8)** @printc
  store i32 (double)* @print_float, i32 (double)** @printd
  store i8** (i8**)* @print_string, i8** (i8**)** @prints
  store double 7.000000e+01, double* @target
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%observable* getelementptr (%observable,
%observable* null, i32 1) to i32))
  %__obs = bitcast i8* %malloccall to %observable*
  %__curr_vpp = getelementptr inbounds %observable, %observable* %__obs, i32 0, i32 0
  store i8* bitcast (double* @target to i8*), i8** %__curr_vpp
  store %observable* %__obs, %observable** @"$relative_humidity"
  store double 2.000000e+01, double* @target.1
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (%observable* getelementptr (%observable,
%observable* null, i32 1) to i32))
  %__obs2 = bitcast i8* %malloccall1 to %observable*
  %__curr_vpp3 = getelementptr inbounds %observable, %observable* %__obs2, i32 0, i32 0
  store i8* bitcast (double* @target.1 to i8*), i8** %__curr_vpp3
  store %observable* %__obs2, %observable** @"$temperature"
  store i8 99, i8* @target.2
  %malloccall4 = tail call i8* @malloc(i32 ptrtoint (%observable* getelementptr (%observable,
%observable* null, i32 1) to i32))
  %__obs5 = bitcast i8* %malloccall4 to %observable*
  %__curr_vpp6 = getelementptr inbounds %observable, %observable* %__obs5, i32 0, i32 0
  store i8* @target.2, i8** %__curr_vpp6
  store %observable* %__obs5, %observable** @"$temp_type"
  store double (double, i8)* @user_func, double (double, i8)** @fahrenheit_converter
  %fahrenheit_converter = load double (double, i8)*, double (double, i8)**
@fahrenheit_converter
  %"$temperature" = load %observable*, %observable** @"$temperature"
  %"$temp_type" = load %observable*, %observable** @"$temp_type"
  %malloccall7 = tail call i8* @malloc(i32 ptrtoint (%observable* getelementptr (%observable,
%observable* null, i32 1) to i32))
  %__obs8 = bitcast i8* %malloccall7 to %observable*
  %__curr_vpp9 = getelementptr inbounds %observable, %observable* %__obs8, i32 0, i32 0
  store i8* bitcast (double* @target.3 to i8*), i8** %__curr_vpp9
  store %observable* %__obs8, %observable** @sub_obs
```

```
  %__func = getelementptr inbounds %observable, %observable* %__obs8, i32 0, i32 3
  %func_to_i8 = bitcast double (double, i8)* %fahrenheit_converter to i8*
  store i8* %func_to_i8, i8** %__func
  call void @subscribe(%observable* %"$temperature", %observable* %__obs8)
  call void @subscribe(%observable* %"$temp_type", %observable* %__obs8)
  %__ups1_curr_vpp = getelementptr inbounds %observable, %observable* %"$temperature",
i32 0, i32 0
  %__ups1_curr_vp = load i8*, i8** %__ups1_curr_vpp
  %__dwn_upv1_vpp = getelementptr inbounds %observable, %observable* %__obs8, i32 0, i32
1
  store i8* %__ups1_curr_vp, i8** %__dwn_upv1_vpp
  %__ups2_curr_vpp = getelementptr inbounds %observable, %observable* %"$temp_type", i32
0, i32 0
  %__ups2_curr_vp = load i8*, i8** %__ups2_curr_vpp
  %__dwn_upv2_vpp = getelementptr inbounds %observable, %observable* %__obs8, i32 0, i32
2
  store i8* %__ups2_curr_vp, i8** %__dwn_upv2_vpp
  %__converter = getelementptr inbounds %observable, %observable* %__obs8, i32 0, i32 6
  store void (i8*, i8*, i8*, i8*)* @converter, void (i8*, i8*, i8*, i8*)** %__converter
  %_r = getelementptr inbounds %observable, %observable* %__obs8, i32 0, i32 0
  %_r10 = load i8*, i8** %_r
  %_p1 = getelementptr inbounds %observable, %observable* %__obs8, i32 0, i32 1
  %_p111 = load i8*, i8** %_p1
  %_p2 = getelementptr inbounds %observable, %observable* %__obs8, i32 0, i32 2
  %_p212 = load i8*, i8** %_p2
  %_f = getelementptr inbounds %observable, %observable* %__obs8, i32 0, i32 3
  %_f13 = load i8*, i8** %_f
  %_c = getelementptr inbounds %observable, %observable* %__obs8, i32 0, i32 6
  %_c14 = load void (i8*, i8*, i8*, i8*)*, void (i8*, i8*, i8*, i8*)** %_c
  call void %_c14(i8* %_r10, i8* %_p111, i8* %_p212, i8* %_f13)
  call void @onNext(%observable* %__obs8)
  store %observable* %__obs8, %observable** @"$t"
  %"$t" = load %observable*, %observable** @"$t"
  %"$relative_humidity" = load %observable*, %observable** @"$relative_humidity"
  %malloccall15 = tail call i8* @malloc(i32 ptrtoint (%observable* getelementptr (%observable,
%observable* null, i32 1) to i32))
```

```
  %__obs16 = bitcast i8* %malloccall15 to %observable*
  %__curr_vpp17 = getelementptr inbounds %observable, %observable* %__obs16, i32 0, i32 0
  store i8* bitcast (double* @target.7 to i8*), i8** %__curr_vpp17
  store %observable* %__obs16, %observable** @sub_obs.8
  %__func18 = getelementptr inbounds %observable, %observable* %__obs16, i32 0, i32 3
  store i8* bitcast (double (double)* @user_func.6 to i8*), i8** %__func18
  call void @subscribe(%observable* %"$relative_humidity", %observable* %__obs16)
  %__ups_curr_vpp = getelementptr inbounds %observable, %observable*
%"$relative_humidity", i32 0, i32 0
  %__ups_curr_vp = load i8*, i8** %__ups_curr_vpp
  %__dwn_upv_vpp = getelementptr inbounds %observable, %observable* %__obs16, i32 0, i32
1
  store i8* %__ups_curr_vp, i8** %__dwn_upv_vpp
  %__converter19 = getelementptr inbounds %observable, %observable* %__obs16, i32 0, i32 6
  store void (i8*, i8*, i8*, i8*)* @converter.9, void (i8*, i8*, i8*, i8*)** %__converter19
  %_r20 = getelementptr inbounds %observable, %observable* %__obs16, i32 0, i32 0
  %_r21 = load i8*, i8** %_r20
  %_p122 = getelementptr inbounds %observable, %observable* %__obs16, i32 0, i32 1
  %_p123 = load i8*, i8** %_p122
  %_f24 = getelementptr inbounds %observable, %observable* %__obs16, i32 0, i32 3
  %_f25 = load i8*, i8** %_f24
  %_c26 = getelementptr inbounds %observable, %observable* %__obs16, i32 0, i32 6
  %_c27 = load void (i8*, i8*, i8*, i8*)*, void (i8*, i8*, i8*, i8*)** %_c26
  call void %_c27(i8* %_r21, i8* %_p123, i8* %_p123, i8* %_f25)
  call void @onNext(%observable* %__obs16)
  %malloccall28 = tail call i8* @malloc(i32 ptrtoint (%observable* getelementptr (%observable,
%observable* null, i32 1) to i32))
  %__obs29 = bitcast i8* %malloccall28 to %observable*
  %__curr_vpp30 = getelementptr inbounds %observable, %observable* %__obs29, i32 0, i32 0
  store i8* bitcast (double* @target.10 to i8*), i8** %__curr_vpp30
  store %observable* %__obs29, %observable** @sub_obs.11
  %__func31 = getelementptr inbounds %observable, %observable* %__obs29, i32 0, i32 3
  store i8* bitcast (double (double)* @user_func.5 to i8*), i8** %__func31
  call void @subscribe(%observable* %__obs16, %observable* %__obs29)
  %__ups_curr_vpp32 = getelementptr inbounds %observable, %observable* %__obs16, i32 0,
i32 0
```

```
  %__ups_curr_vp33 = load i8*, i8** %__ups_curr_vpp32
  %__dwn_upv_vpp34 = getelementptr inbounds %observable, %observable* %__obs29, i32 0,
i32 1
  store i8* %__ups_curr_vp33, i8** %__dwn_upv_vpp34
  %__converter35 = getelementptr inbounds %observable, %observable* %__obs29, i32 0, i32 6
  store void (i8*, i8*, i8*, i8*)* @converter.12, void (i8*, i8*, i8*, i8*)** %__converter35
  %_r36 = getelementptr inbounds %observable, %observable* %__obs29, i32 0, i32 0
  %_r37 = load i8*, i8** %_r36
  %_p138 = getelementptr inbounds %observable, %observable* %__obs29, i32 0, i32 1
  %_p139 = load i8*, i8** %_p138
  %_f40 = getelementptr inbounds %observable, %observable* %__obs29, i32 0, i32 3
  %_f41 = load i8*, i8** %_f40
  %_c42 = getelementptr inbounds %observable, %observable* %__obs29, i32 0, i32 6
  %_c43 = load void (i8*, i8*, i8*, i8*)*, void (i8*, i8*, i8*, i8*)** %_c42
  call void %_c43(i8* %_r37, i8* %_p139, i8* %_p139, i8* %_f41)
  call void @onNext(%observable* %__obs29)
  %malloccall44 = tail call i8* @malloc(i32 ptrtoint (%observable* getelementptr (%observable,
%observable* null, i32 1) to i32))
  %__obs45 = bitcast i8* %malloccall44 to %observable*
  %__curr_vpp46 = getelementptr inbounds %observable, %observable* %__obs45, i32 0, i32 0
  store i8* bitcast (double* @target.13 to i8*), i8** %__curr_vpp46
  store %observable* %__obs45, %observable** @sub_obs.14
  %__func47 = getelementptr inbounds %observable, %observable* %__obs45, i32 0, i32 3
  store i8* bitcast (double (double, double)* @user_func.4 to i8*), i8** %__func47
  call void @subscribe(%observable* %"$t", %observable* %__obs45)
  call void @subscribe(%observable* %__obs29, %observable* %__obs45)
  %__ups1_curr_vpp48 = getelementptr inbounds %observable, %observable* %"$t", i32 0, i32
0
  %__ups1_curr_vp49 = load i8*, i8** %__ups1_curr_vpp48
  %__dwn_upv1_vpp50 = getelementptr inbounds %observable, %observable* %__obs45, i32 0,
i32 1
  store i8* %__ups1_curr_vp49, i8** %__dwn_upv1_vpp50
  %__ups2_curr_vpp51 = getelementptr inbounds %observable, %observable* %__obs29, i32 0,
i32 0
  %__ups2_curr_vp52 = load i8*, i8** %__ups2_curr_vpp51
```

```
  %__dwn_upv2_vpp53 = getelementptr inbounds %observable, %observable* %__obs45, i32 0,
i32 2
  store i8* %__ups2_curr_vp52, i8** %__dwn_upv2_vpp53
  %__converter54 = getelementptr inbounds %observable, %observable* %__obs45, i32 0, i32 6
  store void (i8*, i8*, i8*, i8*)* @converter.15, void (i8*, i8*, i8*, i8*)** %__converter54
  %_r55 = getelementptr inbounds %observable, %observable* %__obs45, i32 0, i32 0
  %_r56 = load i8*, i8** %_r55
  %_p157 = getelementptr inbounds %observable, %observable* %__obs45, i32 0, i32 1
  %_p158 = load i8*, i8** %_p157
  %_p259 = getelementptr inbounds %observable, %observable* %__obs45, i32 0, i32 2
  %_p260 = load i8*, i8** %_p259
  %_f61 = getelementptr inbounds %observable, %observable* %__obs45, i32 0, i32 3
  %_f62 = load i8*, i8** %_f61
  %_c63 = getelementptr inbounds %observable, %observable* %__obs45, i32 0, i32 6
  %_c64 = load void (i8*, i8*, i8*, i8*)*, void (i8*, i8*, i8*, i8*)** %_c63
  call void %_c64(i8* %_r56, i8* %_p158, i8* %_p260, i8* %_f62)
  call void @onNext(%observable* %__obs45)
  store %observable* %__obs45, %observable** @"$out"
  %"$out" = load %observable*, %observable** @"$out"
  %malloccall65 = tail call i8* @malloc(i32 ptrtoint (%observable* getelementptr (%observable,
%observable* null, i32 1) to i32))
  %__obs66 = bitcast i8* %malloccall65 to %observable*
  %__curr_vpp67 = getelementptr inbounds %observable, %observable* %__obs66, i32 0, i32 0
  store i8* @target.17, i8** %__curr_vpp67
  store %observable* %__obs66, %observable** @sub_obs.18
  %__func68 = getelementptr inbounds %observable, %observable* %__obs66, i32 0, i32 3
  store i8* bitcast (void (double)* @user_func.16 to i8*), i8** %__func68
  %__ups_curr_vpp69 = getelementptr inbounds %observable, %observable* %"$out", i32 0, i32
0
  %__ups_curr_vp70 = load i8*, i8** %__ups_curr_vpp69
  %__dwn_upv_vpp71 = getelementptr inbounds %observable, %observable* %__obs66, i32 0,
i32 1
  store i8* %__ups_curr_vp70, i8** %__dwn_upv_vpp71
  call void @subscribe(%observable* %"$out", %observable* %__obs66)
  %__converter72 = getelementptr inbounds %observable, %observable* %__obs66, i32 0, i32 6
  store void (i8*, i8*, i8*, i8*)* @converter.19, void (i8*, i8*, i8*, i8*)** %__converter72
```

```
  %_r73 = getelementptr inbounds %observable, %observable* %__obs66, i32 0, i32 0
  %_r74 = load i8*, i8** %_r73
  %_p175 = getelementptr inbounds %observable, %observable* %__obs66, i32 0, i32 1
  %_p176 = load i8*, i8** %_p175
  %_f77 = getelementptr inbounds %observable, %observable* %__obs66, i32 0, i32 3
  %_f78 = load i8*, i8** %_f77
  %_c79 = getelementptr inbounds %observable, %observable* %__obs66, i32 0, i32 6
  %_c80 = load void (i8*, i8*, i8*, i8*)*, void (i8*, i8*, i8*, i8*)** %_c79
  call void %_c80(i8* %_r74, i8* %_p176, i8* %_p176, i8* %_f78)
  call void @onNext(%observable* %__obs66)
  %"$relative_humidity81" = load %observable*, %observable** @"$relative_humidity"
  %__curr_vpp82 = getelementptr inbounds %observable, %observable*
%"$relative_humidity81", i32 0, i32 0
  %__curr_vp = load i8*, i8** %__curr_vpp82
  %i8_to_curr = bitcast i8* %__curr_vp to double*
  store double 6.000000e+01, double* %i8_to_curr
  call void @onNext(%observable* %"$relative_humidity81")
  %"$relative_humidity83" = load %observable*, %observable** @"$relative_humidity"
  %__curr_vpp84 = getelementptr inbounds %observable, %observable*
%"$relative_humidity83", i32 0, i32 0
  %__curr_vp85 = load i8*, i8** %__curr_vpp84
  %i8_to_curr86 = bitcast i8* %__curr_vp85 to double*
  store double 5.000000e+01, double* %i8_to_curr86
  call void @onNext(%observable* %"$relative_humidity83")
  %"$temperature87" = load %observable*, %observable** @"$temperature"
  %__curr_vpp88 = getelementptr inbounds %observable, %observable* %"$temperature87", i32
0, i32 0
  %__curr_vp89 = load i8*, i8** %__curr_vpp88
  %i8_to_curr90 = bitcast i8* %__curr_vp89 to double*
  store double 7.000000e+01, double* %i8_to_curr90
  call void @onNext(%observable* %"$temperature87")
  %"$temp_type91" = load %observable*, %observable** @"$temp_type"
  %__curr_vpp92 = getelementptr inbounds %observable, %observable* %"$temp_type91", i32
0, i32 0
  %__curr_vp93 = load i8*, i8** %__curr_vpp92
  store i8 102, i8* %__curr_vp93
```

```
  call void @onNext(%observable* %"$temp_type91")
  ret i32 0
}

declare i32 @print_int(i32)

declare i32 @print_char(i8)

declare i32 @print_float(double)

declare i8** @print_string(i8**)

declare i8** @array_concat_int(i8**, i8**, i8**)

declare i8** @array_concat_char(i8**, i8**, i8**)

define void @onNext(%observable* %upstream) {
entry:
  %upstream1 = alloca %observable*
  store %observable* %upstream, %observable** %upstream1
  %__ups = load %observable*, %observable** %upstream1
  %__sub_pp = getelementptr inbounds %observable, %observable* %__ups, i32 0, i32 4
  %__sub_p = load %subscription*, %subscription** %__sub_pp
  %current_p = alloca %subscription*
  store %subscription* %__sub_p, %subscription** %current_p
  br label %while_pred

while_pred:                                       ; preds = %while_body, %entry
  %current = load %subscription*, %subscription** %current_p
  %tmp = icmp ne %subscription* null, %current
  br i1 %tmp, label %while_body, label %merge

while_body:                                       ; preds = %while_pred
  %__d_ref = getelementptr inbounds %subscription, %subscription* %current, i32 0, i32 1
  %__d = load %observable*, %observable** %__d_ref
  %__next_p = getelementptr inbounds %subscription, %subscription* %current, i32 0, i32 0
```

```
  %__next = load %subscription*, %subscription** %__next_p
  store %subscription* %__next, %subscription** %current_p
  %_r = getelementptr inbounds %observable, %observable* %__d, i32 0, i32 0
  %_r2 = load i8*, i8** %_r
  %_p1 = getelementptr inbounds %observable, %observable* %__d, i32 0, i32 1
  %_p13 = load i8*, i8** %_p1
  %_p2 = getelementptr inbounds %observable, %observable* %__d, i32 0, i32 2
  %_p24 = load i8*, i8** %_p2
  %_f = getelementptr inbounds %observable, %observable* %__d, i32 0, i32 3
  %_f5 = load i8*, i8** %_f
  %_c = getelementptr inbounds %observable, %observable* %__d, i32 0, i32 6
  %_c6 = load void (i8*, i8*, i8*, i8*)*, void (i8*, i8*, i8*, i8*)** %_c
  call void %_c6(i8* %_r2, i8* %_p13, i8* %_p24, i8* %_f5)
  call void @onNext(%observable* %__d)
  br label %while_pred

merge:                                            ; preds = %while_pred
  ret void
}

define void @subscribe(%observable* %upstream, %observable* %downstream) {
entry:
  %upstream1 = alloca %observable*
  %downstream2 = alloca %observable*
  store %observable* %upstream, %observable** %upstream1
  store %observable* %downstream, %observable** %downstream2
  %__ups = load %observable*, %observable** %upstream1
  %__dws = load %observable*, %observable** %downstream2
  %__subscription_pp = getelementptr inbounds %observable, %observable* %__ups, i32 0, i32
4
  %__subscription_p = load %subscription*, %subscription** %__subscription_pp
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr
(i1*, i1** null, i32 1) to i64), i64 2) to i32))
  %new = bitcast i8* %malloccall to %subscription*
  %__next = getelementptr inbounds %subscription, %subscription* %new, i32 0, i32 0
  %__obs = getelementptr inbounds %subscription, %subscription* %new, i32 0, i32 1
```

```
  store %subscription* %__subscription_p, %subscription** %__next
  store %observable* %__dws, %observable** %__obs
  store %subscription* %new, %subscription** %__subscription_pp
  ret void
}

define void @complete(%observable* %upstream) {
entry:
  %upstream1 = alloca %observable*
  store %observable* %upstream, %observable** %upstream1
  %__ups = load %observable*, %observable** %upstream1
  %__subscription_pp = getelementptr inbounds %observable, %observable* %__ups, i32 0, i32
4
  store %subscription* null, %subscription** %__subscription_pp
  ret void
}

declare noalias i8* @malloc(i32)

define double @user_func(double %temp1, i8 %type2) {
entry:
  %temp = alloca double
  store double %temp1, double* %temp
  %type = alloca i8
  store i8 %type2, i8* %type
  %adjusted = alloca double
  %type3 = load i8, i8* %type
  %tmp = icmp eq i8 %type3, 99
  br i1 %tmp, label %then, label %else

merge:                                            ; preds = %else, %then
  %adjusted10 = load double, double* %adjusted
  ret double %adjusted10

then:                                             ; preds = %entry
  %adjusted5 = load double, double* %temp
```

```
  store double %adjusted5, double* %adjusted
  br label %merge

else:                                             ; preds = %entry
  %temp6 = load double, double* %temp
  %tmp7 = fsub double %temp6, 3.200000e+01
  %adjusted9 = fdiv double %tmp7, 1.800000e+00
  store double %adjusted9, double* %adjusted
  br label %merge
}

define void @converter(i8* %r, i8* %p1, i8* %p2, i8* %f) {
entry:
  %i8_to_r = bitcast i8* %r to double*
  %i8_to_p1 = bitcast i8* %p1 to double*
  %i8_to_f = bitcast i8* %f to double (double, i8)*
  %p11 = load double, double* %i8_to_p1
  %p22 = load i8, i8* %p2
  %result = call double %i8_to_f(double %p11, i8 %p22)
  store double %result, double* %i8_to_r
  ret void
}

define double @user_func.4(double %x1, double %y2) {
entry:
  %x = alloca double
  store double %x1, double* %x
  %y = alloca double
  store double %y2, double* %y
  %x3 = load double, double* %x
  %y4 = load double, double* %y
  %tmp = fsub double %x3, %y4
  ret double %tmp
}

define double @user_func.5(double %x1) {
```

```
entry:
  %x = alloca double
  store double %x1, double* %x
  %x2 = load double, double* %x
  %tmp = fdiv double %x2, 5.000000e+00
  ret double %tmp
}

define double @user_func.6(double %x1) {
entry:
  %x = alloca double
  store double %x1, double* %x
  %x2 = load double, double* %x
  %tmp = fsub double 1.000000e+02, %x2
  ret double %tmp
}

define void @converter.9(i8* %r, i8* %p1, i8* %p2, i8* %f) {
entry:
  %i8_to_r = bitcast i8* %r to double*
  %i8_to_p1 = bitcast i8* %p1 to double*
  %i8_to_f = bitcast i8* %f to double (double)*
  %p11 = load double, double* %i8_to_p1
  %result = call double %i8_to_f(double %p11)
  store double %result, double* %i8_to_r
  ret void
}

define void @converter.12(i8* %r, i8* %p1, i8* %p2, i8* %f) {
entry:
  %i8_to_r = bitcast i8* %r to double*
  %i8_to_p1 = bitcast i8* %p1 to double*
  %i8_to_f = bitcast i8* %f to double (double)*
  %p11 = load double, double* %i8_to_p1
  %result = call double %i8_to_f(double %p11)
  store double %result, double* %i8_to_r
```

```
  ret void
}

define void @converter.15(i8* %r, i8* %p1, i8* %p2, i8* %f) {
entry:
  %i8_to_r = bitcast i8* %r to double*
  %i8_to_p1 = bitcast i8* %p1 to double*
  %i8_to_p2 = bitcast i8* %p2 to double*
  %i8_to_f = bitcast i8* %f to double (double, double)*
  %p11 = load double, double* %i8_to_p1
  %p22 = load double, double* %i8_to_p2
  %result = call double %i8_to_f(double %p11, double %p22)
  store double %result, double* %i8_to_r
  ret void
}

define void @user_func.16(double %x1) {
entry:
  %x = alloca double
  store double %x1, double* %x
  %prints = load i8** (i8**)*, i8** (i8**)** @prints
  %a = alloca i8*, i32 15
  %e = getelementptr i8*, i8** %a, i32 0
  %p = bitcast i8** %e to i32**
  %ealloc = alloca i32
  store i32 14, i32* %ealloc
  store i32* %ealloc, i32** %p
  %e2 = getelementptr i8*, i8** %a, i32 1
  %ealloc3 = alloca i8
  store i8 78, i8* %ealloc3
  store i8* %ealloc3, i8** %e2
  %e4 = getelementptr i8*, i8** %a, i32 2
  %ealloc5 = alloca i8
  store i8 101, i8* %ealloc5
  store i8* %ealloc5, i8** %e4
  %e6 = getelementptr i8*, i8** %a, i32 3
```

```
%ealloc7 = alloca i8
store i8 119, i8* %ealloc7
store i8* %ealloc7, i8** %e6
%e8 = getelementptr i8*, i8** %a, i32 4
%ealloc9 = alloca i8
store i8 32, i8* %ealloc9
store i8* %ealloc9, i8** %e8
%e10 = getelementptr i8*, i8** %a, i32 5
%ealloc11 = alloca i8
store i8 100, i8* %ealloc11
store i8* %ealloc11, i8** %e10
%e12 = getelementptr i8*, i8** %a, i32 6
%ealloc13 = alloca i8
store i8 101, i8* %ealloc13
store i8* %ealloc13, i8** %e12
%e14 = getelementptr i8*, i8** %a, i32 7
%ealloc15 = alloca i8
store i8 119, i8* %ealloc15
store i8* %ealloc15, i8** %e14
%e16 = getelementptr i8*, i8** %a, i32 8
%ealloc17 = alloca i8
store i8 32, i8* %ealloc17
store i8* %ealloc17, i8** %e16
%e18 = getelementptr i8*, i8** %a, i32 9
%ealloc19 = alloca i8
store i8 112, i8* %ealloc19
store i8* %ealloc19, i8** %e18
%e20 = getelementptr i8*, i8** %a, i32 10
%ealloc21 = alloca i8
store i8 111, i8* %ealloc21
store i8* %ealloc21, i8** %e20
%e22 = getelementptr i8*, i8** %a, i32 11
%ealloc23 = alloca i8
store i8 105, i8* %ealloc23
store i8* %ealloc23, i8** %e22
%e24 = getelementptr i8*, i8** %a, i32 12
```

```
  %ealloc25 = alloca i8
  store i8 110, i8* %ealloc25
  store i8* %ealloc25, i8** %e24
  %e26 = getelementptr i8*, i8** %a, i32 13
  %ealloc27 = alloca i8
  store i8 116, i8* %ealloc27
  store i8* %ealloc27, i8** %e26
  %e28 = getelementptr i8*, i8** %a, i32 14
  %ealloc29 = alloca i8
  store i8 58, i8* %ealloc29
  store i8* %ealloc29, i8** %e28
  %0 = call i8** %prints(i8** %a)
  %printf = load i32 (double)*, i32 (double)** @printd
  %x30 = load double, double* %x
  %f_result = call i32 %printf(double %x30)
  %prints31 = load i8** (i8**)*, i8** (i8**)** @prints
  %a32 = alloca i8*
  %e33 = getelementptr i8*, i8** %a32, i32 0
  %p34 = bitcast i8** %e33 to i32**
  %ealloc35 = alloca i32
  store i32 0, i32* %ealloc35
  store i32* %ealloc35, i32** %p34
  %1 = call i8** %prints31(i8** %a32)
  ret void
}

define void @converter.19(i8* %r, i8* %p1, i8* %p2, i8* %f) {
entry:
  %i8_to_p1 = bitcast i8* %p1 to double*
  %i8_to_f = bitcast i8* %f to i8 (double)*
  %p11 = load double, double* %i8_to_p1
  %0 = call i8 %i8_to_f(double %p11)
  ret void
}
```

Source program 2:

```
int fib(int x) {
  int y = if (x > 2) fib(x-1) + fib(x-2) else 1;
  return y;
}

prints("6th fibonacci number:");
printi(fib(6));

struct Math {
  int a;
  float b;
  char[] verb;
  (int, float)->(float) op;
};

struct Math m = {4, 3.14, "times", (int x, float y)->{ return x * y; }}; /* Anonymous functions */
struct Math d = {10, 3.5, "divided by", (int x, float y)->{ return x / y; }};

void eval(struct Math x, char[] operation) {
  prints("Performing " + operation);
  printi(x.a);
  prints(x.verb);
  printf(x.b);
  prints("equals");
  printf((x.op)(x.a, x.b));
  return;
}

eval(m, "multiplication");
eval(d, "division");
```

Target result 2:

```
; ModuleID = 'Seaflow'
source_filename = "Seaflow"

%observable = type { i8*, i8*, i8*, i8*, %subscription*, i32, void (i8*, i8*, i8*, i8*)* }
%subscription = type { %subscription*, %observable* }

@printi = global i32 (i32)* null
@printc = global i32 (i8)* null
@printd = global i32 (double)* null
@prints = global i8** (i8**)* null
@fib = global i32 (i32)* null
@m = global { i32, double, i8**, double (i32, double)* }* null
@d = global { i32, double, i8**, double (i32, double)* }* null
@eval = global void ({ i32, double, i8**, double (i32, double)* }*, i8**)* null

define i32 @main() {
entry:
  store i32 (i32)* @print_int, i32 (i32)** @printi
  store i32 (i8)* @print_char, i32 (i8)** @printc
  store i32 (double)* @print_float, i32 (double)** @printd
  store i8** (i8**)* @print_string, i8** (i8**)** @prints
  store i32 (i32)* @user_func, i32 (i32)** @fib
  %prints = load i8** (i8**)*, i8** (i8**)** @prints
  %a = alloca i8*, i32 22
  %e = getelementptr i8*, i8** %a, i32 0
  %p = bitcast i8** %e to i32**
  %ealloc = alloca i32
  store i32 21, i32* %ealloc
  store i32* %ealloc, i32** %p
  %e1 = getelementptr i8*, i8** %a, i32 1
  %ealloc2 = alloca i8
  store i8 54, i8* %ealloc2
  store i8* %ealloc2, i8** %e1
  %e3 = getelementptr i8*, i8** %a, i32 2
  %ealloc4 = alloca i8
  store i8 116, i8* %ealloc4
```

```
store i8* %ealloc4, i8** %e3
%e5 = getelementptr i8*, i8** %a, i32 3
%ealloc6 = alloca i8
store i8 104, i8* %ealloc6
store i8* %ealloc6, i8** %e5
%e7 = getelementptr i8*, i8** %a, i32 4
%ealloc8 = alloca i8
store i8 32, i8* %ealloc8
store i8* %ealloc8, i8** %e7
%e9 = getelementptr i8*, i8** %a, i32 5
%ealloc10 = alloca i8
store i8 102, i8* %ealloc10
store i8* %ealloc10, i8** %e9
%e11 = getelementptr i8*, i8** %a, i32 6
%ealloc12 = alloca i8
store i8 105, i8* %ealloc12
store i8* %ealloc12, i8** %e11
%e13 = getelementptr i8*, i8** %a, i32 7
%ealloc14 = alloca i8
store i8 98, i8* %ealloc14
store i8* %ealloc14, i8** %e13
%e15 = getelementptr i8*, i8** %a, i32 8
%ealloc16 = alloca i8
store i8 111, i8* %ealloc16
store i8* %ealloc16, i8** %e15
%e17 = getelementptr i8*, i8** %a, i32 9
%ealloc18 = alloca i8
store i8 110, i8* %ealloc18
store i8* %ealloc18, i8** %e17
%e19 = getelementptr i8*, i8** %a, i32 10
%ealloc20 = alloca i8
store i8 97, i8* %ealloc20
store i8* %ealloc20, i8** %e19
%e21 = getelementptr i8*, i8** %a, i32 11
%ealloc22 = alloca i8
store i8 99, i8* %ealloc22
```

```
store i8* %ealloc22, i8** %e21
%e23 = getelementptr i8*, i8** %a, i32 12
%ealloc24 = alloca i8
store i8 99, i8* %ealloc24
store i8* %ealloc24, i8** %e23
%e25 = getelementptr i8*, i8** %a, i32 13
%ealloc26 = alloca i8
store i8 105, i8* %ealloc26
store i8* %ealloc26, i8** %e25
%e27 = getelementptr i8*, i8** %a, i32 14
%ealloc28 = alloca i8
store i8 32, i8* %ealloc28
store i8* %ealloc28, i8** %e27
%e29 = getelementptr i8*, i8** %a, i32 15
%ealloc30 = alloca i8
store i8 110, i8* %ealloc30
store i8* %ealloc30, i8** %e29
%e31 = getelementptr i8*, i8** %a, i32 16
%ealloc32 = alloca i8
store i8 117, i8* %ealloc32
store i8* %ealloc32, i8** %e31
%e33 = getelementptr i8*, i8** %a, i32 17
%ealloc34 = alloca i8
store i8 109, i8* %ealloc34
store i8* %ealloc34, i8** %e33
%e35 = getelementptr i8*, i8** %a, i32 18
%ealloc36 = alloca i8
store i8 98, i8* %ealloc36
store i8* %ealloc36, i8** %e35
%e37 = getelementptr i8*, i8** %a, i32 19
%ealloc38 = alloca i8
store i8 101, i8* %ealloc38
store i8* %ealloc38, i8** %e37
%e39 = getelementptr i8*, i8** %a, i32 20
%ealloc40 = alloca i8
store i8 114, i8* %ealloc40
```

```
store i8* %ealloc40, i8** %e39
%e41 = getelementptr i8*, i8** %a, i32 21
%ealloc42 = alloca i8
store i8 58, i8* %ealloc42
store i8* %ealloc42, i8** %e41
%0 = call i8** %prints(i8** %a)
%printi = load i32 (i32)*, i32 (i32)** @printi
%fib = load i32 (i32)*, i32 (i32)** @fib
%f_result = call i32 %fib(i32 6)
%f_result43 = call i32 %printi(i32 %f_result)
%a44 = alloca i8*, i32 6
%e45 = getelementptr i8*, i8** %a44, i32 0
%p46 = bitcast i8** %e45 to i32**
%ealloc47 = alloca i32
store i32 5, i32* %ealloc47
store i32* %ealloc47, i32** %p46
%e48 = getelementptr i8*, i8** %a44, i32 1
%ealloc49 = alloca i8
store i8 116, i8* %ealloc49
store i8* %ealloc49, i8** %e48
%e50 = getelementptr i8*, i8** %a44, i32 2
%ealloc51 = alloca i8
store i8 105, i8* %ealloc51
store i8* %ealloc51, i8** %e50
%e52 = getelementptr i8*, i8** %a44, i32 3
%ealloc53 = alloca i8
store i8 109, i8* %ealloc53
store i8* %ealloc53, i8** %e52
%e54 = getelementptr i8*, i8** %a44, i32 4
%ealloc55 = alloca i8
store i8 101, i8* %ealloc55
store i8* %ealloc55, i8** %e54
%e56 = getelementptr i8*, i8** %a44, i32 5
%ealloc57 = alloca i8
store i8 115, i8* %ealloc57
store i8* %ealloc57, i8** %e56
```

```
  %struct = alloca { i32, double, i8**, double (i32, double)* }
  %e58 = getelementptr inbounds { i32, double, i8**, double (i32, double)* }, { i32, double,
i8**, double (i32, double)* }* %struct, i32 0, i32 0
  store i32 4, i32* %e58
  %e59 = getelementptr inbounds { i32, double, i8**, double (i32, double)* }, { i32, double,
i8**, double (i32, double)* }* %struct, i32 0, i32 1
  store double 3.140000e+00, double* %e59
  %e60 = getelementptr inbounds { i32, double, i8**, double (i32, double)* }, { i32, double,
i8**, double (i32, double)* }* %struct, i32 0, i32 2
  store i8** %a44, i8*** %e60
  %e61 = getelementptr inbounds { i32, double, i8**, double (i32, double)* }, { i32, double,
i8**, double (i32, double)* }* %struct, i32 0, i32 3
  store double (i32, double)* @user_func.1, double (i32, double)** %e61
  store { i32, double, i8**, double (i32, double)* }* %struct, { i32, double, i8**, double (i32,
double)* }** @m
  %a62 = alloca i8*, i32 11
  %e63 = getelementptr i8*, i8** %a62, i32 0
  %p64 = bitcast i8** %e63 to i32**
  %ealloc65 = alloca i32
  store i32 10, i32* %ealloc65
  store i32* %ealloc65, i32** %p64
  %e66 = getelementptr i8*, i8** %a62, i32 1
  %ealloc67 = alloca i8
  store i8 100, i8* %ealloc67
  store i8* %ealloc67, i8** %e66
  %e68 = getelementptr i8*, i8** %a62, i32 2
  %ealloc69 = alloca i8
  store i8 105, i8* %ealloc69
  store i8* %ealloc69, i8** %e68
  %e70 = getelementptr i8*, i8** %a62, i32 3
  %ealloc71 = alloca i8
  store i8 118, i8* %ealloc71
  store i8* %ealloc71, i8** %e70
  %e72 = getelementptr i8*, i8** %a62, i32 4
  %ealloc73 = alloca i8
  store i8 105, i8* %ealloc73
```

```
  store i8* %ealloc73, i8** %e72
  %e74 = getelementptr i8*, i8** %a62, i32 5
  %ealloc75 = alloca i8
  store i8 100, i8* %ealloc75
  store i8* %ealloc75, i8** %e74
  %e76 = getelementptr i8*, i8** %a62, i32 6
  %ealloc77 = alloca i8
  store i8 101, i8* %ealloc77
  store i8* %ealloc77, i8** %e76
  %e78 = getelementptr i8*, i8** %a62, i32 7
  %ealloc79 = alloca i8
  store i8 100, i8* %ealloc79
  store i8* %ealloc79, i8** %e78
  %e80 = getelementptr i8*, i8** %a62, i32 8
  %ealloc81 = alloca i8
  store i8 32, i8* %ealloc81
  store i8* %ealloc81, i8** %e80
  %e82 = getelementptr i8*, i8** %a62, i32 9
  %ealloc83 = alloca i8
  store i8 98, i8* %ealloc83
  store i8* %ealloc83, i8** %e82
  %e84 = getelementptr i8*, i8** %a62, i32 10
  %ealloc85 = alloca i8
  store i8 121, i8* %ealloc85
  store i8* %ealloc85, i8** %e84
  %struct86 = alloca { i32, double, i8**, double (i32, double)* }
  %e87 = getelementptr inbounds { i32, double, i8**, double (i32, double)* }, { i32, double,
i8**, double (i32, double)* }* %struct86, i32 0, i32 0
  store i32 10, i32* %e87
  %e88 = getelementptr inbounds { i32, double, i8**, double (i32, double)* }, { i32, double,
i8**, double (i32, double)* }* %struct86, i32 0, i32 1
  store double 3.500000e+00, double* %e88
  %e89 = getelementptr inbounds { i32, double, i8**, double (i32, double)* }, { i32, double,
i8**, double (i32, double)* }* %struct86, i32 0, i32 2
  store i8** %a62, i8*** %e89
```

```
  %e90 = getelementptr inbounds { i32, double, i8**, double (i32, double)* }, { i32, double, i8**, double (i32, double)* }* %struct86, i32 0, i32 3
  store double (i32, double)* @user_func.2, double (i32, double)** %e90
  store { i32, double, i8**, double (i32, double)* }* %struct86, { i32, double, i8**, double (i32, double)* }** @d
  store void ({ i32, double, i8**, double (i32, double)* }*, i8**)* @user_func.3, void ({ i32, double, i8**, double (i32, double)* }*, i8**)** @eval
  %eval = load void ({ i32, double, i8**, double (i32, double)* }*, i8**)*, void ({ i32, double, i8**, double (i32, double)* }*, i8**)** @eval
  %a91 = alloca i8*, i32 15
  %e92 = getelementptr i8*, i8** %a91, i32 0
  %p93 = bitcast i8** %e92 to i32**
  %ealloc94 = alloca i32
  store i32 14, i32* %ealloc94
  store i32* %ealloc94, i32** %p93
  %e95 = getelementptr i8*, i8** %a91, i32 1
  %ealloc96 = alloca i8
  store i8 109, i8* %ealloc96
  store i8* %ealloc96, i8** %e95
  %e97 = getelementptr i8*, i8** %a91, i32 2
  %ealloc98 = alloca i8
  store i8 117, i8* %ealloc98
  store i8* %ealloc98, i8** %e97
  %e99 = getelementptr i8*, i8** %a91, i32 3
  %ealloc100 = alloca i8
  store i8 108, i8* %ealloc100
  store i8* %ealloc100, i8** %e99
  %e101 = getelementptr i8*, i8** %a91, i32 4
  %ealloc102 = alloca i8
  store i8 116, i8* %ealloc102
  store i8* %ealloc102, i8** %e101
  %e103 = getelementptr i8*, i8** %a91, i32 5
  %ealloc104 = alloca i8
  store i8 105, i8* %ealloc104
  store i8* %ealloc104, i8** %e103
  %e105 = getelementptr i8*, i8** %a91, i32 6
```

```
%ealloc106 = alloca i8
store i8 112, i8* %ealloc106
store i8* %ealloc106, i8** %e105
%e107 = getelementptr i8*, i8** %a91, i32 7
%ealloc108 = alloca i8
store i8 108, i8* %ealloc108
store i8* %ealloc108, i8** %e107
%e109 = getelementptr i8*, i8** %a91, i32 8
%ealloc110 = alloca i8
store i8 105, i8* %ealloc110
store i8* %ealloc110, i8** %e109
%e111 = getelementptr i8*, i8** %a91, i32 9
%ealloc112 = alloca i8
store i8 99, i8* %ealloc112
store i8* %ealloc112, i8** %e111
%e113 = getelementptr i8*, i8** %a91, i32 10
%ealloc114 = alloca i8
store i8 97, i8* %ealloc114
store i8* %ealloc114, i8** %e113
%e115 = getelementptr i8*, i8** %a91, i32 11
%ealloc116 = alloca i8
store i8 116, i8* %ealloc116
store i8* %ealloc116, i8** %e115
%e117 = getelementptr i8*, i8** %a91, i32 12
%ealloc118 = alloca i8
store i8 105, i8* %ealloc118
store i8* %ealloc118, i8** %e117
%e119 = getelementptr i8*, i8** %a91, i32 13
%ealloc120 = alloca i8
store i8 111, i8* %ealloc120
store i8* %ealloc120, i8** %e119
%e121 = getelementptr i8*, i8** %a91, i32 14
%ealloc122 = alloca i8
store i8 110, i8* %ealloc122
store i8* %ealloc122, i8** %e121
```

```
  %m = load { i32, double, i8**, double (i32, double)* }*, { i32, double, i8**, double (i32, double)* }** @m
  call void %eval({ i32, double, i8**, double (i32, double)* }* %m, i8** %a91)
  %eval123 = load void ({ i32, double, i8**, double (i32, double)* }*, i8**)*, void ({ i32, double, i8**, double (i32, double)* }*, i8**)** @eval
  %a124 = alloca i8*, i32 9
  %e125 = getelementptr i8*, i8** %a124, i32 0
  %p126 = bitcast i8** %e125 to i32**
  %ealloc127 = alloca i32
  store i32 8, i32* %ealloc127
  store i32* %ealloc127, i32** %p126
  %e128 = getelementptr i8*, i8** %a124, i32 1
  %ealloc129 = alloca i8
  store i8 100, i8* %ealloc129
  store i8* %ealloc129, i8** %e128
  %e130 = getelementptr i8*, i8** %a124, i32 2
  %ealloc131 = alloca i8
  store i8 105, i8* %ealloc131
  store i8* %ealloc131, i8** %e130
  %e132 = getelementptr i8*, i8** %a124, i32 3
  %ealloc133 = alloca i8
  store i8 118, i8* %ealloc133
  store i8* %ealloc133, i8** %e132
  %e134 = getelementptr i8*, i8** %a124, i32 4
  %ealloc135 = alloca i8
  store i8 105, i8* %ealloc135
  store i8* %ealloc135, i8** %e134
  %e136 = getelementptr i8*, i8** %a124, i32 5
  %ealloc137 = alloca i8
  store i8 115, i8* %ealloc137
  store i8* %ealloc137, i8** %e136
  %e138 = getelementptr i8*, i8** %a124, i32 6
  %ealloc139 = alloca i8
  store i8 105, i8* %ealloc139
  store i8* %ealloc139, i8** %e138
  %e140 = getelementptr i8*, i8** %a124, i32 7
```

```
  %ealloc141 = alloca i8
  store i8 111, i8* %ealloc141
  store i8* %ealloc141, i8** %e140
  %e142 = getelementptr i8*, i8** %a124, i32 8
  %ealloc143 = alloca i8
  store i8 110, i8* %ealloc143
  store i8* %ealloc143, i8** %e142
  %d = load { i32, double, i8**, double (i32, double)* }*, { i32, double, i8**, double (i32,
double)* }** @d
  call void %eval123({ i32, double, i8**, double (i32, double)* }* %d, i8** %a124)
  ret i32 0
}

declare i32 @print_int(i32)

declare i32 @print_char(i8)

declare i32 @print_float(double)

declare i8** @print_string(i8**)

declare i8** @array_concat_int(i8**, i8**, i8**)

declare i8** @array_concat_char(i8**, i8**, i8**)

define void @onNext(%observable* %upstream) {
entry:
  %upstream1 = alloca %observable*
  store %observable* %upstream, %observable** %upstream1
  %__ups = load %observable*, %observable** %upstream1
  %__sub_pp = getelementptr inbounds %observable, %observable* %__ups, i32 0, i32 4
  %__sub_p = load %subscription*, %subscription** %__sub_pp
  %current_p = alloca %subscription*
  store %subscription* %__sub_p, %subscription** %current_p
  br label %while_pred
```

```
while_pred:                                       ; preds = %while_body, %entry
  %current = load %subscription*, %subscription** %current_p
  %tmp = icmp ne %subscription* null, %current
  br i1 %tmp, label %while_body, label %merge

while_body:                                       ; preds = %while_pred
  %__d_ref = getelementptr inbounds %subscription, %subscription* %current, i32 0, i32 1
  %__d = load %observable*, %observable** %__d_ref
  %__next_p = getelementptr inbounds %subscription, %subscription* %current, i32 0, i32 0
  %__next = load %subscription*, %subscription** %__next_p
  store %subscription* %__next, %subscription** %current_p
  %_r = getelementptr inbounds %observable, %observable* %__d, i32 0, i32 0
  %_r2 = load i8*, i8** %_r
  %_p1 = getelementptr inbounds %observable, %observable* %__d, i32 0, i32 1
  %_p13 = load i8*, i8** %_p1
  %_p2 = getelementptr inbounds %observable, %observable* %__d, i32 0, i32 2
  %_p24 = load i8*, i8** %_p2
  %_f = getelementptr inbounds %observable, %observable* %__d, i32 0, i32 3
  %_f5 = load i8*, i8** %_f
  %_c = getelementptr inbounds %observable, %observable* %__d, i32 0, i32 6
  %_c6 = load void (i8*, i8*, i8*, i8*)*, void (i8*, i8*, i8*, i8*)** %_c
  call void %_c6(i8* %_r2, i8* %_p13, i8* %_p24, i8* %_f5)
  call void @onNext(%observable* %__d)
  br label %while_pred

merge:                                            ; preds = %while_pred
  ret void
}

define void @subscribe(%observable* %upstream, %observable* %downstream) {
entry:
  %upstream1 = alloca %observable*
  %downstream2 = alloca %observable*
  store %observable* %upstream, %observable** %upstream1
  store %observable* %downstream, %observable** %downstream2
  %__ups = load %observable*, %observable** %upstream1
```

```
  %__dws = load %observable*, %observable** %downstream2
  %__subscription_pp = getelementptr inbounds %observable, %observable* %__ups, i32 0, i32
4
  %__subscription_p = load %subscription*, %subscription** %__subscription_pp
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1** getelementptr
(i1*, i1** null, i32 1) to i64), i64 2) to i32))
  %new = bitcast i8* %malloccall to %subscription*
  %__next = getelementptr inbounds %subscription, %subscription* %new, i32 0, i32 0
  %__obs = getelementptr inbounds %subscription, %subscription* %new, i32 0, i32 1
  store %subscription* %__subscription_p, %subscription** %__next
  store %observable* %__dws, %observable** %__obs
  store %subscription* %new, %subscription** %__subscription_pp
  ret void
}

define void @complete(%observable* %upstream) {
entry:
  %upstream1 = alloca %observable*
  store %observable* %upstream, %observable** %upstream1
  %__ups = load %observable*, %observable** %upstream1
  %__subscription_pp = getelementptr inbounds %observable, %observable* %__ups, i32 0, i32
4
  store %subscription* null, %subscription** %__subscription_pp
  ret void
}

define i32 @user_func(i32 %x1) {
entry:
  %x = alloca i32
  store i32 %x1, i32* %x
  %y = alloca i32
  %x2 = load i32, i32* %x
  %tmp = icmp sgt i32 %x2, 2
  br i1 %tmp, label %then, label %else

merge:                                            ; preds = %else, %then
```

```
  %y11 = load i32, i32* %y
  ret i32 %y11

then:                                             ; preds = %entry
  %fib = load i32 (i32)*, i32 (i32)** @fib
  %x3 = load i32, i32* %x
  %tmp4 = sub i32 %x3, 1
  %f_result = call i32 %fib(i32 %tmp4)
  %fib5 = load i32 (i32)*, i32 (i32)** @fib
  %x6 = load i32, i32* %x
  %tmp7 = sub i32 %x6, 2
  %f_result8 = call i32 %fib5(i32 %tmp7)
  %y10 = add i32 %f_result, %f_result8
  store i32 %y10, i32* %y
  br label %merge

else:                                             ; preds = %entry
  store i32 1, i32* %y
  br label %merge
}

define double @user_func.1(i32 %x1, double %y2) {
entry:
  %x = alloca i32
  store i32 %x1, i32* %x
  %y = alloca double
  store double %y2, double* %y
  %x3 = load i32, i32* %x
  %y4 = load double, double* %y
  %tmp = sitofp i32 %x3 to double
  %tmp5 = fmul double %tmp, %y4
  ret double %tmp5
}

define double @user_func.2(i32 %x1, double %y2) {
entry:
```

```
  %x = alloca i32
  store i32 %x1, i32* %x
  %y = alloca double
  store double %y2, double* %y
  %x3 = load i32, i32* %x
  %y4 = load double, double* %y
  %tmp = sitofp i32 %x3 to double
  %tmp5 = fdiv double %tmp, %y4
  ret double %tmp5
}

define void @user_func.3({ i32, double, i8**, double (i32, double)* }* %x1, i8** %operation2)
{
entry:
  %x = alloca { i32, double, i8**, double (i32, double)* }*
  store { i32, double, i8**, double (i32, double)* }* %x1, { i32, double, i8**, double (i32,
double)* }** %x
  %operation = alloca i8**
  store i8** %operation2, i8*** %operation
  %prints = load i8** (i8**)*, i8** (i8**)** @prints
  %a = alloca i8*, i32 12
  %e = getelementptr i8*, i8** %a, i32 0
  %p = bitcast i8** %e to i32**
  %ealloc = alloca i32
  store i32 11, i32* %ealloc
  store i32* %ealloc, i32** %p
  %e3 = getelementptr i8*, i8** %a, i32 1
  %ealloc4 = alloca i8
  store i8 80, i8* %ealloc4
  store i8* %ealloc4, i8** %e3
  %e5 = getelementptr i8*, i8** %a, i32 2
  %ealloc6 = alloca i8
  store i8 101, i8* %ealloc6
  store i8* %ealloc6, i8** %e5
  %e7 = getelementptr i8*, i8** %a, i32 3
  %ealloc8 = alloca i8
```

```
store i8 114, i8* %ealloc8
store i8* %ealloc8, i8** %e7
%e9 = getelementptr i8*, i8** %a, i32 4
%ealloc10 = alloca i8
store i8 102, i8* %ealloc10
store i8* %ealloc10, i8** %e9
%e11 = getelementptr i8*, i8** %a, i32 5
%ealloc12 = alloca i8
store i8 111, i8* %ealloc12
store i8* %ealloc12, i8** %e11
%e13 = getelementptr i8*, i8** %a, i32 6
%ealloc14 = alloca i8
store i8 114, i8* %ealloc14
store i8* %ealloc14, i8** %e13
%e15 = getelementptr i8*, i8** %a, i32 7
%ealloc16 = alloca i8
store i8 109, i8* %ealloc16
store i8* %ealloc16, i8** %e15
%e17 = getelementptr i8*, i8** %a, i32 8
%ealloc18 = alloca i8
store i8 105, i8* %ealloc18
store i8* %ealloc18, i8** %e17
%e19 = getelementptr i8*, i8** %a, i32 9
%ealloc20 = alloca i8
store i8 110, i8* %ealloc20
store i8* %ealloc20, i8** %e19
%e21 = getelementptr i8*, i8** %a, i32 10
%ealloc22 = alloca i8
store i8 103, i8* %ealloc22
store i8* %ealloc22, i8** %e21
%e23 = getelementptr i8*, i8** %a, i32 11
%ealloc24 = alloca i8
store i8 32, i8* %ealloc24
store i8* %ealloc24, i8** %e23
%operation25 = load i8**, i8*** %operation
%ptr = getelementptr i8*, i8** %a, i32 0
```

```
  %ptr26 = getelementptr i8*, i8** %operation25, i32 0
  %0 = getelementptr i8*, i8** %a, i32 0
  %vptr = load i8*, i8** %0
  %iptr = bitcast i8* %vptr to i32*
  %a27 = load i32, i32* %iptr
  %1 = getelementptr i8*, i8** %operation25, i32 0
  %vptr28 = load i8*, i8** %1
  %iptr29 = bitcast i8* %vptr28 to i32*
  %a30 = load i32, i32* %iptr29
  %sum = add i32 %a27, %a30
  %sum2 = add i32 %sum, 1
  %a31 = alloca i8*, i32 %sum2
  %e32 = getelementptr i8*, i8** %a31, i32 0
  %p33 = bitcast i8** %e32 to i32**
  %ealloc34 = alloca i32
  store i32 %sum, i32* %ealloc34
  store i32* %ealloc34, i32** %p33
  %array_concat = call i8** @array_concat_char(i8** %ptr, i8** %ptr26, i8** %a31)
  %2 = call i8** %prints(i8** %a31)
  %printi = load i32 (i32)*, i32 (i32)** @printi
  %x35 = load { i32, double, i8**, double (i32, double)* }*, { i32, double, i8**, double (i32,
double)* }** %x
  %tmp = getelementptr inbounds { i32, double, i8**, double (i32, double)* }, { i32, double,
i8**, double (i32, double)* }* %x35, i32 0, i32 0
  %z = load i32, i32* %tmp
  %f_result = call i32 %printi(i32 %z)
  %prints36 = load i8** (i8**)*, i8** (i8**)** @prints
  %x37 = load { i32, double, i8**, double (i32, double)* }*, { i32, double, i8**, double (i32,
double)* }** %x
  %tmp38 = getelementptr inbounds { i32, double, i8**, double (i32, double)* }, { i32, double,
i8**, double (i32, double)* }* %x37, i32 0, i32 2
  %z39 = load i8**, i8*** %tmp38
  %3 = call i8** %prints36(i8** %z39)
  %printf = load i32 (double)*, i32 (double)** @printd
  %x40 = load { i32, double, i8**, double (i32, double)* }*, { i32, double, i8**, double (i32,
double)* }** %x
```

```
  %tmp41 = getelementptr inbounds { i32, double, i8**, double (i32, double)* }, { i32, double,
i8**, double (i32, double)* }* %x40, i32 0, i32 1
  %z42 = load double, double* %tmp41
  %f_result43 = call i32 %printf(double %z42)
  %prints44 = load i8** (i8**)*, i8** (i8**)** @prints
  %a45 = alloca i8*, i32 7
  %e46 = getelementptr i8*, i8** %a45, i32 0
  %p47 = bitcast i8** %e46 to i32**
  %ealloc48 = alloca i32
  store i32 6, i32* %ealloc48
  store i32* %ealloc48, i32** %p47
  %e49 = getelementptr i8*, i8** %a45, i32 1
  %ealloc50 = alloca i8
  store i8 101, i8* %ealloc50
  store i8* %ealloc50, i8** %e49
  %e51 = getelementptr i8*, i8** %a45, i32 2
  %ealloc52 = alloca i8
  store i8 113, i8* %ealloc52
  store i8* %ealloc52, i8** %e51
  %e53 = getelementptr i8*, i8** %a45, i32 3
  %ealloc54 = alloca i8
  store i8 117, i8* %ealloc54
  store i8* %ealloc54, i8** %e53
  %e55 = getelementptr i8*, i8** %a45, i32 4
  %ealloc56 = alloca i8
  store i8 97, i8* %ealloc56
  store i8* %ealloc56, i8** %e55
  %e57 = getelementptr i8*, i8** %a45, i32 5
  %ealloc58 = alloca i8
  store i8 108, i8* %ealloc58
  store i8* %ealloc58, i8** %e57
  %e59 = getelementptr i8*, i8** %a45, i32 6
  %ealloc60 = alloca i8
  store i8 115, i8* %ealloc60
  store i8* %ealloc60, i8** %e59
  %4 = call i8** %prints44(i8** %a45)
```

```
  %printf61 = load i32 (double)*, i32 (double)** @printd
  %x62 = load { i32, double, i8**, double (i32, double)* }*, { i32, double, i8**, double (i32, double)* }** %x
  %tmp63 = getelementptr inbounds { i32, double, i8**, double (i32, double)* }, { i32, double, i8**, double (i32, double)* }* %x62, i32 0, i32 3
  %z64 = load double (i32, double)*, double (i32, double)** %tmp63
  %x65 = load { i32, double, i8**, double (i32, double)* }*, { i32, double, i8**, double (i32, double)* }** %x
  %tmp66 = getelementptr inbounds { i32, double, i8**, double (i32, double)* }, { i32, double, i8**, double (i32, double)* }* %x65, i32 0, i32 1
  %z67 = load double, double* %tmp66
  %x68 = load { i32, double, i8**, double (i32, double)* }*, { i32, double, i8**, double (i32, double)* }** %x
  %tmp69 = getelementptr inbounds { i32, double, i8**, double (i32, double)* }, { i32, double, i8**, double (i32, double)* }* %x68, i32 0, i32 0
  %z70 = load i32, i32* %tmp69
  %f_result71 = call double %z64(i32 %z70, double %z67)
  %f_result72 = call i32 %printf61(double %f_result71)
  ret void
}

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

6.2 Test Suites

Tests are organized by feature and are located within the parent “test” directory. The test cases for each feature are located within directories that are named to correspond with the particular feature they test. This grouping allows for test cases to easily be included or excluded when running tests, which is helpful when focusing on a particular component. The test suite is split up by the following features: arrays, char, float, functions, global variables, higher-order functions, if statements, int, local variables, observables, and struct. File names for positive test cases follow the naming convention “test-*.flo” with “test-*.out” containing the expected output. Likewise, negative test cases follow the naming convention “fail-*.flo” with “fail-*.err” containing the expected error message. In particular, the negative examples were key in ensuring that unintended behavior was caught and resulted in descriptive error messages.

6.3 Methodology Behind Choosing Test Cases

This section describes the reasoning behind how test cases were chosen for each of the aforementioned features.

- array:
    - Test cases confirmed the creation of arrays of various types and indexing using a variety of methods, including the use of literals, identifiers, and function calls. Tests were created to ensure that functions could return values sourced from local arrays, as well as from arrays passed in as arguments to functions. Additional functionality was also tested including the copying of arrays of various types, array length, and array concatenation. Negative test cases ensured proper type checking when sourcing values from arrays and for creation of arrays, and confirming that a previously defined array cannot be reassigned. An interesting test case that is worth noting is the possibility to create an array of higher-order functions, and subsequently calling a function by indexing into the array.

- char
    - Test cases confirmed the assignment of characters at the global and local level, as well as the use of all valid operators. Negative test cases ensured that previously defined characters could not be reassigned, and that the invalid use of the “&&” and “||” operators on characters were caught.

- float
    - Test cases confirmed the assignment of floats at the global and local level, as well as the use of all valid operators. Additional tests were written to confirm implicit conversion when using operators where one value was a float and the other an integer. Negative test cases ensured that previously defined floats could not be reassigned, and that the invalid use of the “&&” and “||” operators on floats were caught.

- functions
    - Test cases confirmed the creation of functions with various return types including void, varying number of formal arguments and types of formals, use of

formal arguments within functions, assignment of variables to the result of function calls, recursion, the use of the return statement in void functions and the possibility to omit the return statement. The ability to call functions with literals, identifiers, and the result of other function calls was also confirmed. Negative test cases ensured that duplicate functions with the same name, duplicate formal names within the same function, void formals, and void locals are disallowed. Additionally, negative test cases confirmed that function calls with too few or too many arguments were caught, as well as instances where the type of actuals did not match the types of the formals, and the disallowance of empty function bodies. Negative test cases also ensured type checking when assigning variables to the result of function calls.

- global variables
    - Test cases confirmed the creation of global variables of various types. Negative test cases confirmed that the creation of void global variables and redeclaring an already assigned global variable is disallowed.

- higher-order functions
    - Test cases confirmed the ability to declare higher-order functions, pass in higher-order functions into another higher-order function, local higher-order functions within higher-order functions, declaring a higher-order function inline and applying an argument directly, and defining structs with higher-order functions. Negative test cases confirmed the inability to redeclare an already defined function as a higher-order function.

- if statements
    - Test cases confirmed that the appropriate expression is assigned to the variable being set when the condition is true and the condition is false. Additionally, function calls that evaluate to a value used as the expressions for then and else were confirmed to only evaluate the side that is being assigned. Negative test cases confirmed the disallowance of if statements at the global level and ensuring proper type checking when assigning values resulting from an if statement.

- int

- Test cases confirmed the assignment of integers at the global and local level, as well as the use of all valid operators. Negative test cases ensured proper type checking when assigning variables to integer values and that previously defined integers could not be reassigned.

- local variables
  - Test cases confirmed the creation of local variables of various types. Negative test cases confirmed that the creation of void local variables and redeclaring an already assigned local variable is disallowed.

- observables
  - Test cases confirmed the ability to use subscribe, map, combine, and complete on observables, and ensured that when updating the value of observables that all downstream observables were updated appropriately. Multiple subscriptions and chaining of observables were thoroughly tested, as well as the use of overloaded mathematical operators. Negative test cases ensured that only functions with the correct formal argument types could subscribe to an observable, and that subscribe, map, combine, and complete would not accept primitive types as an argument, or when supplied with the incorrect number of arguments.

- struct
  - Test cases confirmed the ability to declare structs containing a variety of types, passing in structs to functions, assigning a struct to a previously defined struct with the same types, the ability to pass a struct into a function that takes a different struct type as its argument but with the same types, and the use of struct arrays. Negative test cases ensured that struct definitions could not be defined in the local scope.

6.4 Test Automation

With 116 test cases in total, test automation was essential in order to speed up the development process. The script “testall.sh” was used to run all tests within the “test” directory. This script uses the filename of the test file to determine whether it is a positive test case (prepended with “test-”) or a negative test case (prepended with “fail-”). After compiling and running each test, the output of positive test cases are compared to the corresponding output file, and the error output of negative test cases are compared to the corresponding error file. Running the command

“make test” compiles and runs all tests, with passing cases marked as “OK” and failing test cases marked with “FAILED.” The results of each test case are output to the file “testall.log”. Note: depending on the machine used to run tests, parsing errors may have different outputs and cause tests to fail.

7 Lessons Learned

7.1 Rohan Arora

This project was definitely a huge learning experience, and I have several takeaways. I’ve always been a bit hesitant to ask for advice or help, but working with this team has really taught me to utilize the strengths of my team members to clear up my uncertainties. My advice is to ask questions early and often from your team members, not only will it help get you up to speed, but it often leads to important discussions which are crucial for the development of the team and project. I also learned the importance of having a comprehensive test environment, it would be impossible to debug without it. Before starting on a new feature make sure to have as many test cases written as you can think of. Definitely don’t stop there though: iteratively add more as you think of cases that could break your code. Always run all test cases, you never know when a change could break a feature that was working fine previously! Finally, I learned the importance of semantic checking. This stage in the pipeline is crucial for code generation, and it is just as important to include descriptive error messages wherever possible.

7.2 Junyang Jin

Coding with other group members has been one of the few pleasures I have experienced this semester. VSCode live share is a great tool for groups that are planning for remote code collaborations. OCaml sometimes allows the source code to compile with warning and conflicts, but it is always a good idea to listen to OCaml and fix stuff. My favorite part of the project is the builder in codegen.ml. Needs to do some digging in LLVM documents to understand what to do.

Advice: magic link: https://llvm.moe/ocaml/Llvm.html

7.3 Sanlok Ho Lee

I think having not just one plan but several plans that we can fall back to helped us a lot. Our final language is a bit different from what we first proposed but I think things went mostly within

what we initially predicted and we were able to deliver most of the key features that we proposed. Another lesson that I learned is to fully understand the code and language syntax before using it. Initially I somewhat blindly copied some MicroC code over to our source code without fully understanding it and later we found out that things do not work well with our design. We still have a lot of duplicated, unused code in our source code, but I think the source code could have been much cleaner if we started after we had better understanding of Ocaml and MicroC source code.

Setting up the development environment correctly as early as possible was a huge factor in successful collaboration. We all setup our environment on docker, created github repository, and installed VSCode CodeShare since the beginning of the project, and we faced almost no technical issue and everyone was able to focus on project implementation itself only.

Thanks for the project, I have a better understanding of language designs and how well (or badly) existing languages are designed. Learning and working with OCaml was really great. Although I have some experience in programming in functional style but it is my first time actually learning a functional programming language I am glad that I got the chance to learn it.

7.4 Sarah Seidman

The biggest thing I learned working on this project is how important it is to envision everything you will want to do with a feature before you implement it. I learned this the hard way implementing arrays (and to a lesser extent structs). I was in such a hurry to get the basic functionality working that I didn't realize that the way I was going about things would make it almost impossible to support other features later on, and I ended up having to rethink my approach more than once.

I also found that sometimes the best way to solve a problem is to turn off your computer and go for a walk. This is true of all programming and problem-solving activities, but I found it especially true working with OCaml and its LLVM library, both of which were new to me and finicky in ways I wasn't used to. Trying to continue when you're frustrated is a great way to make the task take longer.

8 Appendix

8.1 Complete Code Listing

8.1.1 Source Code

8.1.1.1 ./Dockerfile

```
# Based on 20.04 LTS
FROM ubuntu:focal

ARG DEBIAN_FRONTEND=noninteractive
ENV TZ=America/New_York

RUN apt-get -yq update && \
    apt-get -y upgrade && \
    apt-get -yq --no-install-suggests --no-install-recommends install \
    ocaml \
    menhir \
    llvm-10 \
    llvm-10-dev \
    m4 \
    git \
    aspcud \
    ca-certificates \
    python2.7 \
    pkg-config \
    cmake \
    opam

RUN ln -s /usr/bin/lli-10 /usr/bin/lli
RUN ln -s /usr/bin/llc-10 /usr/bin/llc

RUN opam init --disable-sandboxing
RUN opam install --yes llvm.10.0.0
RUN opam install --yes ocamlfind

RUN apt-get install ocamlbuild

WORKDIR /root
```

```
ENTRYPOINT ["opam", "config", "exec", "--"]

CMD ["bash"]
```

8.1.1.2 ./Makefile

```
# "make test" Compiles everything and runs the regression tests

.PHONY : test
test : all testall.sh
	./testall.sh

# "make all" builds the executable as well as the "utils" library for array
concatenation

.PHONY : all
all : seaflow.native utils.o

# "make seaflow.native" compiles the compiler
#
# The _tags file controls the operation of ocamlbuild, e.g., by including
# packages, enabling warnings
#
# See https://github.com/ocaml/ocamlbuild/blob/master/manual/manual.adoc

seaflow.native :
	opam config exec -- \
	ocamlbuild -use-ocamlfind seaflow.native

# "make clean" removes all generated files

.PHONY : clean
clean :
	ocamlbuild -clean
	rm -rf testall.log ocamlllvm *.diff
	rm -f *.cmo
	rm -f *.cmi
```

```
	rm -f seaflowparse.ml
	rm -f seaflowparse.mli
	rm -f seaflowparse.output
	rm -f *.o

# Building the array concatenation util

utils : utils.c
	cc -o utils utils.c

# Building the tarball

TESTS = \
  basic1 declarations

FAILS = \
  decl

TESTFILES = $(TESTS:%=test-%.flo) $(TESTS:%=test-%.out) \
	    $(FAILS:%=fail-%.flo) $(FAILS:%=fail-%.err)

TARFILES = ast.ml sast.ml codegen.ml Makefile _tags seaflow.ml seaflowparse.mly \
	scanner.mll semant.ml testall.sh \
	utils.c arcade-font.pbm font2c \
	Dockerfile \
	$(TESTFILES:%=test_seaflow/%)

seaflow.tar.gz : $(TARFILES)
	cd .. && tar czf proj/seaflow.tar.gz \
	      $(TARFILES:%=proj/%)
```

8.1.1.3 ./_tags

```
# Include the llvm and llvm.analysis packages while compiling
true: package(llvm), package(llvm.analysis)

# Enable almost all compiler warnings
true : warn(+a-4)
```

```
# Instruct ocamlbuild to ignore the "utils.o" file when it's building
"utils.o": not_hygienic
```

8.1.1.4 ./ast.ml

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

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

type uop = Neg

type typ = Int | Float | Void | Char | Arr of typ | Struct of string | Sbody of
typ list | Func of typ list * typ
  | Observable of typ | Bool

(* type otyp = Observable of typ *)


type bind = typ * string

type expr =
    Literal of int
  | Bliteral of bool
  | Fliteral of string
  | Chliteral of char
  | Strliteral of string
  | Aliteral of expr list
  | Id of string
  | Sid of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Call of expr * expr list
  | Ref of expr * string
  | Arr_Ref of expr * expr
  | FuncExpr of bind list * stmt list
```

```
  | Sliteral of expr list
  | Noexpr
  | Null

and stmt =
    Block of stmt list
  | Expr of expr
  | If of typ * string * expr * expr * expr
  | Return of expr
  | Print of expr
  | Decl of typ * string * expr
  | Str_Def of string * bind list


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


type oexpr =
  | OId of string
  | OBinop1 of oexpr * op * expr   (* For map operation *)
  | OBinop2 of expr * op * oexpr
  | OBinop3 of oexpr * op * oexpr  (* For combine operation *)
  | OUnop of uop * oexpr
  | Map of expr * oexpr
  | Combine of expr * oexpr * oexpr


type obs_stmt =
    Obs of typ * string
  | OExpr of oexpr
  | ODecl of typ * string * expr
  | OODecl of typ * string * oexpr
  | OAssign of string * expr
  | OOAssign of string * oexpr
  | OArr_Decl of typ * string * expr list
```

```
  | OStr_Decl of typ * string * expr list
  | Subscribe of expr * oexpr
  | Complete of oexpr

type glob =
    Stmt of stmt
  | Fdecl of func_decl
  | Obs_Stmt of obs_stmt

type program = glob list


(* Pretty-printing functions *)

let rec string_of_typ = function
    Int -> "int"
  | Float -> "float"
  | Void -> "void"
  | Char -> "char"
  | Arr(typ) -> string_of_typ typ ^ "[]"
  | Struct(str) -> "struct " ^ str
  | Sbody(tlist) -> "{ " ^ String.concat ", " (List.map string_of_typ tlist) ^ "
}"
  | Func(typ_list, typ) -> "(" ^
      String.concat ", " (List.map string_of_typ typ_list)
      ^ ") -> (" ^ string_of_typ typ ^ ")"
  | Observable(typ) -> string_of_typ typ ^ "$"
  | Bool -> "bool"

let typ_of_arr = function
  | Arr(typ) -> typ
  | _ -> raise (Failure ("Not an array"))

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

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

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

let string_of_uop = function
    Neg -> "-"

let rec string_of_expr = function
    Literal(l) -> string_of_int l
  | Bliteral(b) -> if b then "TRUE" else "FALSE"
  | Fliteral(l) -> l
  | Chliteral(l) -> "'" ^ String.make 1 l ^ "'"
  | Aliteral(l) -> "[" ^ String.concat "," (List.map string_of_expr l) ^ "]"
  | Strliteral(s) -> "\"" ^ s ^ "\""
  | Id(s) -> s
  | Sid(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
  | Call(ef, el) ->
    string_of_expr ef ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^
")"
  | Ref(e, s) -> string_of_expr e ^ "." ^ s
  | Arr_Ref(e1, e2) -> string_of_expr e1 ^ "[" ^ string_of_expr e2 ^ "]"
  | FuncExpr(bind_list, stmt_list) -> "(" ^
      String.concat ", " (List.map string_of_bind bind_list) ^ ") -> {" ^
      String.concat "" (List.map string_of_stmt stmt_list) ^ "}"
  | Sliteral(e_list) -> "{ " ^ String.concat ", " (List.map string_of_expr
e_list) ^ " }"
  | Noexpr -> ""
  | Null -> ""

and
```

```
string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
  | Print(expr) -> "print(" ^ string_of_expr expr ^ ");\n"
  | Decl(t, s, expr) -> string_of_typ t ^ " " ^ s ^ " = "^ string_of_expr expr ^
";\n"
  | Str_Def(s, bind_list) -> "struct " ^ s ^ " { " ^
      String.concat "\n" (List.map string_of_bind bind_list) ^ "\n};\n"
  | If(t, s, e1, e2, e3) -> string_of_typ t ^ " " ^ s ^ " = if(" ^ string_of_expr
e1 ^ ") "
      ^ string_of_expr e2 ^ " else " ^ string_of_expr e3


let rec string_of_oexpr = function
    OId(s) -> s
  | OBinop1(e1, o, e2) ->
    string_of_oexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
  | OBinop2(e1, o, e2) ->
    string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_oexpr e2
  | OBinop3(e1, o, e2) ->
    string_of_oexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_oexpr e2
  | OUnop(o, e) -> string_of_uop o ^ string_of_oexpr e
  | Map(e, oe) -> "map(" ^ string_of_expr e ^ ", " ^ string_of_oexpr oe ^ ")"
  | Combine(e, oe1, oe2) -> "combine(" ^ string_of_expr e ^ ", " ^
      string_of_oexpr oe1 ^ ", " ^ string_of_oexpr oe2 ^ ")"


let string_of_obs_stmt = function
    Obs(t, s) -> string_of_typ t ^ " " ^ s ^ ";\n"
  | OExpr(e) -> string_of_oexpr e ^ ";\n"
  | ODecl(t, s, e) -> string_of_typ t ^ " " ^ s ^ " = " ^ string_of_expr e ^
";\n"
  | OODecl(t, s, e) -> string_of_typ t ^ " " ^ s ^ " = " ^ string_of_oexpr e ^
";\n"
  | OAssign(s, e) -> s ^ " = " ^ string_of_expr e ^ ";\n"
  | OOAssign(s, e) -> s ^ " = " ^ string_of_oexpr e ^ ";\n"
  | OArr_Decl(t, s, expr_list) -> string_of_typ t ^ " " ^ s ^ " = [" ^
      String.concat ", " (List.map string_of_expr expr_list) ^ "];\n"
```

```
  | OStr_Decl(t, s, expr_list) -> string_of_typ t ^ " " ^ s ^ " = {" ^
      String.concat ", " (List.map string_of_expr expr_list) ^ "};\n"
  | Subscribe(e, oe) ->
      "subscribe(" ^ string_of_expr e ^ ", " ^ string_of_oexpr oe ^ ");\n"
  | Complete(oe) ->
      "complete(" ^ string_of_oexpr oe ^ ");\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 str_of_glob = function
  Stmt(stmt) -> string_of_stmt stmt
  | Fdecl(func_decl) -> string_of_fdecl func_decl
  | Obs_Stmt(obs_stmt) -> string_of_obs_stmt obs_stmt

let string_of_program (globs) =
  String.concat "" (List.map str_of_glob globs) ^ "\n"
```

8.1.1.5 ./codegen.ml

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

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

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

Detailed documentation on the OCaml LLVM library:

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

```
*)

module L = Llvm
module A = Ast
open Sast
module StringHash = Hashtbl.Make(struct
  type t = string
  let equal x y = x = y
  let hash = Hashtbl.hash
end)


type observable_class = SingleObservable | DoubleObservable

let global_vars : L.llvalue StringHash.t = StringHash.create 10
let global_structs = StringHash.create 10
(* must keep list of types to know what to cast to when doing array reference *)
let arr_types = StringHash.create 10

(* translate : Sast.program -> Llvm.module *)
let translate (globs) =
  let context    = L.global_context () in

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

  (* Get types from the context *)
  let i32_t      = L.i32_type    context
  and i8_t       = L.i8_type     context
  and i1_t       = L.i1_type     context
  and float_t    = L.double_type context
  (* and char_t     = L.i8_type     context *)
  and void_t     = L.void_type   context
  and void_ptr_t = L.pointer_type (L.i8_type context)
  and struct_t   = L.struct_type context     in

  (* function to convert bind list to array of lltypes *)
  let rec list_to_lltype l = Array.of_list (List.map ltype_of_typ l)
```

```
(* Return the LLVM type for a Seaflow type *)
and ltype_of_typ : A.typ -> L.lltype = function
    A.Int   -> i32_t
  | A.Char  -> i8_t
  | A.Bool  -> i1_t
  | A.Float -> float_t
  | A.Void  -> void_t
  | A.Arr(_) -> L.pointer_type void_ptr_t
  | A.Struct(str) ->
    let (_, tlist, _) = try (StringHash.find global_structs ("struct " ^ str))
      with Not_found -> raise (Failure "Struct type mismatch")
    in
    let t = struct_t (Array.of_list (List.map ltype_of_typ tlist)) in
    L.pointer_type t
  | A.Func(param_types, rtype) ->
    let param_ltypes = (List.map ltype_of_typ param_types) in
    let rltype = ltype_of_typ rtype in
    L.pointer_type (L.function_type rltype (Array.of_list param_ltypes))
  | A.Observable _ -> L.pointer_type obv_t
  | _ -> raise (Failure "Unknown type 2200")


and obv_t = L.named_struct_type context "observable" in
let obv_pt = L.pointer_type obv_t in
let subscription_t  = L.named_struct_type context "subscription" in
let subscription_pt = L.pointer_type subscription_t in
let converter_t = L.pointer_type (L.function_type void_t
  ([| void_ptr_t; void_ptr_t; void_ptr_t; void_ptr_t |])) in

ignore(L.struct_set_body obv_t [|
  void_ptr_t;       (* pointer to current value *)
  void_ptr_t;       (* pointer to upstream value 1 *)
  void_ptr_t;       (* pointer to upstream value 2 *)
  void_ptr_t;       (* pointer to function *)
  subscription_pt;  (* pointer to subscription linked list *)
  i32_t;            (* observable type 1 if single, 2 if double *)
  converter_t       (* pointer to the converter *)
|] false);

ignore(L.struct_set_body subscription_t [|
```

```
    subscription_pt;  (* pointer to next subscription *)
    obv_pt            (* pointer to observable *)
  |] false);


  let main_ftype = L.function_type i32_t [| |] in
  let main_function = L.define_function "main" main_ftype the_module in

  let global_builder = L.builder_at_end context (L.entry_block main_function) in



  let printi_t : L.lltype =
    L.function_type i32_t [| i32_t |] in
  let printf_t : L.lltype =
    L.function_type i32_t [| float_t |] in
  let printc_t : L.lltype =
    L.function_type i32_t [| i8_t |] in
  let print_string_t : L.lltype =
    L.function_type (L.pointer_type void_ptr_t) [| L.pointer_type void_ptr_t |]
in

  let _ : L.llvalue =
    let f = L.declare_function "print_int" printi_t the_module in
    let store = L.define_global "printi" (L.const_null (L.pointer_type printi_t))
the_module in
    ignore(L.build_store f store global_builder);
    ignore(StringHash.add global_vars "printi" store);
    f
  in

  let _ : L.llvalue =
    let f = L.declare_function "print_char" printc_t the_module in
    let store = L.define_global "printc" (L.const_null (L.pointer_type printc_t))
the_module in
    ignore(L.build_store f store global_builder);
    ignore(StringHash.add global_vars "printc" store);
    f
  in
```

```
  let _ : L.llvalue =
    let f = L.declare_function "print_float" printf_t the_module in
    let store = L.define_global "printd" (L.const_null (L.pointer_type printf_t))
the_module in
    ignore(L.build_store f store global_builder);
    ignore(StringHash.add global_vars "printf" store);
    f
  in

  let _ : L.llvalue =
      let f = L.declare_function "print_string" print_string_t the_module in
      let store = L.define_global "prints" (L.const_null (L.pointer_type
print_string_t)) the_module in
      ignore(L.build_store f store global_builder);
      ignore(StringHash.add global_vars "prints" store);
      f
  in


  let array_concat_t : L.lltype =
    L.function_type (L.pointer_type void_ptr_t)
        [| L.pointer_type void_ptr_t ; L.pointer_type void_ptr_t ; L.pointer_type
void_ptr_t |] in
  let array_concat_func : L.llvalue =
      L.declare_function "array_concat" array_concat_t the_module in

  let get_base (t: A.typ) = match t with
      A.Float -> L.const_float float_t 0.0
    | A.Int   -> L.const_int i32_t 0
    | A.Bool  -> L.const_int i1_t 0
    | A.Char  -> L.const_int i8_t 0
    | A.Arr(ty) -> L.const_null (ltype_of_typ ty)
    | A.Func(_, _) as f -> L.const_null (ltype_of_typ f)
    | A.Struct(_) as s -> L.const_pointer_null (ltype_of_typ s)
    | A.Void  -> L.const_int i8_t 0
    | _ as x -> raise (Failure ("No match found for " ^ A.string_of_typ x))
  in

  (* LLVM insists each basic block end with exactly one "terminator"
     instruction that transfers control.  This function runs "instr builder"
```

```
    (* if the current block does not already have a terminator.  Used,
    e.g., to handle the "fall off the end of the function" case. *)
  let add_terminal builder instr =
    match L.block_terminator (L.insertion_block builder) with
      Some _ -> ()
    | None -> ignore (instr builder) in

  let next_ftype = L.function_type void_t [| obv_pt |] in
  let next_function = L.define_function "onNext" next_ftype the_module in
  StringHash.add global_vars "onNext" next_function;
  let subscribe_ftype = L.function_type void_t [| obv_pt; obv_pt |] in
  let subscribe_function = L.define_function "subscribe" subscribe_ftype
the_module in
  StringHash.add global_vars "subscribe" subscribe_function;
  let complete_ftype = L.function_type void_t [| obv_pt |] in
  let complete_function = L.define_function "complete" complete_ftype the_module
in
  StringHash.add global_vars "complete" complete_function;


  let lookup vars n = try StringHash.find vars n
      with Not_found ->
        try StringHash.find global_vars n
        with Not_found -> raise (Failure ("Variable " ^ n ^ " not found"))
  in

  let add_store_arr ptr builder i e =
    let idx = L.const_int i32_t i in
    let eptr = L.build_gep ptr [|idx|] "e" builder in
    let cptr = L.build_pointercast eptr
        (L.pointer_type (L.pointer_type (L.type_of e))) "p" builder in
    let ealloc = L.build_alloca (L.type_of e) "ealloc" builder in
    ignore(L.build_store e ealloc builder);
    ignore(L.build_store ealloc cptr builder); i+1
  in

  let get_array_len a builder =
    let idx = L.const_int i32_t 0 in
    let len = L.build_gep a [| idx |] "" builder in
    let v = L.build_load len "vptr" builder in
```

```
    let iptr = L.build_pointercast v (L.pointer_type i32_t) "iptr" builder in
    L.build_load iptr "a" builder
  in

  let arr_ty_of_expr = function
    | (_, SId(s)) -> ltype_of_typ (StringHash.find arr_types s)
    | (_, SAliteral(t, _)) -> ltype_of_typ t
    | _ -> raise (Failure "Wrong type for arr_ty_of_expr")
  in

  let rec expr vars builder ((tt, e) : sexpr) = match e with
      SLiteral i   -> L.const_int i32_t i
    | SBliteral b  -> L.const_int i1_t (if b then 1 else 0)
    | SChliteral c -> L.const_int i8_t (Char.code c)
    | SFliteral l  -> L.const_float_of_string float_t l
    | SId s        -> L.build_load (lookup vars s) s builder
    | SAliteral (_, a) ->
      let elems = List.map (expr vars builder) a in
      let num = List.length elems in
      let ptr = L.build_array_alloca void_ptr_t (L.const_int i32_t (num+1)) "a"
builder in
      ignore(add_store_arr ptr builder 0 (L.const_int i32_t num));
      ignore(List.fold_left (add_store_arr ptr builder) 1 elems); ptr
    | SRef(str_name, type_of_struct, fieldname) ->
      let loc = L.build_load (lookup vars str_name) str_name builder in
      let (_, tlist, flist) = StringHash.find global_structs type_of_struct in
      let rec find x lst =
        match lst with
        | [] -> raise (Failure "Struct name not found")
        | h :: t -> if x = h then 0 else 1 + find x t
      in
      let idx = find fieldname flist in
      let p = L.build_struct_gep loc idx "tmp" builder in
      let ty' = ltype_of_typ (List.nth tlist idx) in
      let ptr = L.build_pointercast p (L.pointer_type ty') "ptr" builder in
      L.build_load ptr "z" builder
    | SSliteral(expr_list) ->
        let elems = List.map (expr vars builder) expr_list in
        let ty = struct_t (Array.of_list (List.map L.type_of elems)) in
        let ptr = L.build_alloca ty "struct" builder in
```

```
        let add_store i e =
          let eptr = L.build_struct_gep ptr i "e" builder in
          let cptr = L.build_pointercast eptr (L.pointer_type (L.type_of e)) "p"
builder in
          ignore(L.build_store e cptr builder) ; i+1
        in
        ignore(List.fold_left add_store 0 elems) ;
        L.build_pointercast ptr (L.pointer_type ty) "pcast" builder
    | SArr_Ref(e1, e2) ->
      let ty = arr_ty_of_expr e1 in
      let arr = expr vars builder e1 in
      let idx = expr vars builder e2 in
      let sum = L.build_add idx (L.const_int i32_t 1) "sum" builder in
      let vptr = L.build_gep arr [| sum |] "" builder in
      let v = L.build_load vptr "vptr" builder in
      let iptr = L.build_pointercast v (L.pointer_type ty) "iptr" builder in
      L.build_load iptr "a" builder
    | SLen(e) -> get_array_len (expr vars builder e) builder
    | SFuncExpr(params, rt, sstmts) ->
      build_function (params, rt, sstmts)
    | SBinop ((A.Arr(_),_ ) as e1, _, e2) ->
        let arr = expr vars builder e1 in
        let arr2 = expr vars builder e2 in
        let ptr = L.build_gep arr [| L.const_int i32_t 0 |] "ptr" builder in
        let ptr2 = L.build_gep arr2 [| L.const_int i32_t 0 |] "ptr" builder in

        let len1 = get_array_len arr builder in
        let len2 = get_array_len arr2 builder in
        let total_len = L.build_add len1 len2 "sum" builder in
        let make_len = L.build_add total_len (L.const_int i32_t 1) "sum2" builder
in

        let ptr3 = L.build_array_alloca void_ptr_t make_len "a" builder in
        ignore(add_store_arr ptr3 builder 0 total_len) ;
        ignore(L.build_call array_concat_func [| ptr ; ptr2 ; ptr3 |]
"array_concat" builder);
        ptr3

    | SBinop (e1, op, e2) ->
      let e1' = expr vars builder e1
```

```
      and e2' = expr vars builder e2 in
      let t1 = L.type_of e1'
      and t2 = L.type_of e2' in
      let same = t1 = t2 in
      let same_int_or_char = (same && (t1 = i32_t || t1 = i8_t))
      and same_float = (same && t1 = float_t)
      and same_bool = (same && t1 = i1_t)
      and not_same = (not(same) && (t1 = i32_t || t1 = float_t) && (t2 = i32_t ||
t2 = float_t))
      and float_left = (t1 = float_t && t2 = i32_t)
      and float_right = (t1 = i32_t && t2 = float_t) in
      let to_float v = L.build_sitofp v float_t "tmp" builder in
      (match op with
        A.Add when same_int_or_char       -> L.build_add e1' e2' "tmp" builder
      | A.Sub when same_int_or_char       -> L.build_sub e1' e2' "tmp" builder
      | A.Mult when same_int_or_char      -> L.build_mul e1' e2' "tmp" builder
      | A.Div when same_int_or_char       -> L.build_sdiv e1' e2' "tmp" builder
      | A.Equal when same_int_or_char     -> L.build_icmp L.Icmp.Eq e1' e2' "tmp"
builder
      | A.Neq when same_int_or_char       -> L.build_icmp L.Icmp.Ne e1' e2' "tmp"
builder
      | A.Less when same_int_or_char      -> L.build_icmp L.Icmp.Slt e1' e2'
"tmp" builder
      | A.Leq when same_int_or_char       -> L.build_icmp L.Icmp.Sle e1' e2'
"tmp" builder
      | A.Greater when same_int_or_char   -> L.build_icmp L.Icmp.Sgt e1' e2'
"tmp" builder
      | A.Geq when same_int_or_char       -> L.build_icmp L.Icmp.Sge e1' e2'
"tmp" builder

      | A.And when same_bool              -> L.build_and e1' e2' "tmp" builder
      | A.Or when same_bool               -> L.build_or e1' e2' "tmp" builder

      | A.Add when same_float     -> L.build_fadd e1' e2' "tmp" builder
      | A.Sub when same_float     -> L.build_fsub e1' e2' "tmp" builder
      | A.Mult when same_float    -> L.build_fmul e1' e2' "tmp" builder
      | A.Div when same_float     -> L.build_fdiv e1' e2' "tmp" builder
      | A.Equal when same_float   -> L.build_fcmp L.Fcmp.Oeq e1' e2' "tmp"
builder
```

```
      | A.Neq when same_float     -> L.build_fcmp L.Fcmp.One e1' e2' "tmp"
builder
      | A.Less when same_float    -> L.build_fcmp L.Fcmp.Olt e1' e2' "tmp"
builder
      | A.Leq when same_float     -> L.build_fcmp L.Fcmp.Ole e1' e2' "tmp"
builder
      | A.Greater when same_float -> L.build_fcmp L.Fcmp.Ogt e1' e2' "tmp"
builder
      | A.Geq when same_float     -> L.build_fcmp L.Fcmp.Oge e1' e2' "tmp"
builder

      | A.Add when not_same     -> L.build_fadd (to_float e1') (to_float e2')
"tmp" builder
      | A.Sub when not_same     -> L.build_fsub (to_float e1') (to_float e2')
"tmp" builder
      | A.Mult when not_same    -> L.build_fmul (to_float e1') (to_float e2')
"tmp" builder
      | A.Div when not_same     -> L.build_fdiv (to_float e1') (to_float e2')
"tmp" builder
      | A.Equal when not_same   -> L.build_fcmp L.Fcmp.Oeq (to_float e1')
(to_float e2') "tmp" builder
      | A.Neq when not_same     -> L.build_fcmp L.Fcmp.One (to_float e1')
(to_float e2') "tmp" builder
      | A.Less when not_same    -> L.build_fcmp L.Fcmp.Olt (to_float e1')
(to_float e2') "tmp" builder
      | A.Leq when not_same     -> L.build_fcmp L.Fcmp.Ole (to_float e1')
(to_float e2') "tmp" builder
      | A.Greater when not_same -> L.build_fcmp L.Fcmp.Ogt (to_float e1')
(to_float e2') "tmp" builder
      | A.Geq when not_same     -> L.build_fcmp L.Fcmp.Oge (to_float e1')
(to_float e2') "tmp" builder

      | A.And | A.Or when (same_float || float_left || float_right) ->
          raise (Failure "internal error: semant should have rejected and/or on
float")
      | _ -> raise (Failure "Unknown Binop case")
      )
    | SUnop(op, ((t, _) as e)) ->
      let e' = expr vars builder e in
      (match op with
```

```
        A.Neg when t = A.Float -> L.build_fneg
      | A.Neg                  -> L.build_neg) e' "tmp" builder
    | SCall(f, args) ->
      let fdef = expr vars builder f in

      let llargs = List.rev (List.map (expr vars builder) (List.rev args)) in
      let result = (match tt with
                    A.Void -> ""
                  | _ -> "f_result") in

      L.build_call fdef (Array.of_list llargs) result builder
    | _ as x -> raise (Failure ("Not Implemented 2003 " ^ (string_of_sexpr (tt,
x))))

  (* Define each function (arguments and return type) so we can
  call it even before we've created its body *)
  and build_function (params, rt, sstmts) =
    let name = "user_func"
    and formal_types = Array.of_list (List.map (fun (t, _) -> ltype_of_typ t)
params) in

    let ftype = L.function_type (ltype_of_typ rt) formal_types in
    let the_function = L.define_function name ftype the_module in

    let local_builder = L.builder_at_end context (L.entry_block the_function) in

    let local_vars = StringHash.create 10 in


    let _add_store (t, n) = (* Returns a point to local store *)
      let _ = match StringHash.find_opt local_vars n with
        | Some(_) -> raise (Failure "Cannot have duplicate formal!")
        | None -> ()
      in
      let _store = L.build_alloca (ltype_of_typ t) n local_builder in
      let _ = match t with
        | A.Arr(x) -> StringHash.add arr_types n x
        | _ -> ()
      in
      StringHash.add local_vars n _store ;
```

```
  _store
in

let _fill_store v store = (* fills store *)
  ignore (L.build_store v store local_builder); ()
in

let add_formal (t, n) p =
  let local_store = _add_store (t, n) in
  L.set_value_name n p;
  _fill_store p local_store;
  let _ = match t with
    | A.Arr(x) -> StringHash.add arr_types n x
    | _ -> ()
  in
  ()
in

let _ = List.iter2 add_formal params
  (Array.to_list (L.params the_function))
in

let rec build_local_stmt builder = function
    SExpr e -> ignore(expr local_vars builder e); builder
  | SBlock(stmt_list) -> List.fold_left build_local_stmt builder stmt_list
  | SDecl(t, s, e) ->
    let local_store = _add_store (t, s) in
    let e' = expr local_vars builder e in
    _fill_store e' local_store;
    builder
  | SReturn e -> ignore(match rt with
            (* Special "return nothing" instr *)
            A.Void -> L.build_ret_void builder
            (* Build return statement *)
          | _ -> L.build_ret (expr local_vars builder e) builder ); builder
  | SIf(ltyp, var, cond, then_stmt, else_stmt) ->
    let local = L.build_alloca (ltype_of_typ ltyp) var builder in
    let bool_val = expr local_vars builder cond in
    let merge_bb = L.append_block context "merge" the_function in
    let then_bb = L.append_block context "then" the_function in
```

```
      let then_builder = L.builder_at_end context then_bb in
      let then_val = expr local_vars then_builder then_stmt in
      L.set_value_name var then_val;
      ignore(L.build_store then_val local then_builder) ;
      StringHash.add local_vars var local ;
      ignore(L.build_br merge_bb then_builder) ;
      let else_bb = L.append_block context "else" the_function in
      let else_builder = L.builder_at_end context else_bb in
      let else_val = expr local_vars else_builder else_stmt in
      ignore(L.set_value_name var else_val) ;
      ignore(L.build_store else_val local else_builder);
      StringHash.add local_vars var local ;
      ignore(L.build_br merge_bb else_builder) ;
      ignore(L.build_cond_br bool_val then_bb else_bb builder);
      L.builder_at_end context merge_bb
    | _ -> raise (Failure "Not Implemented 2005")
  in

  let local_builder_out = build_local_stmt local_builder (SBlock sstmts) in

  add_terminal local_builder_out (match rt with
          A.Void -> L.build_ret_void
        | A.Float -> L.build_ret (L.const_float float_t 0.0)
        | t -> L.build_ret (L.const_int (ltype_of_typ t) 0));
  the_function
in


let make_observable (t, e) builder =
  let v_store = L.define_global "target" (get_base t) the_module in
  let _ = match e with
    | Some x -> ignore(L.build_store x v_store builder)
    | None -> ()
  in
  let obs = L.build_malloc obv_t "__obs" builder in

  let curr_vpp = L.build_struct_gep obs 0 "__curr_vpp" builder in
  let curr_vp = L.build_bitcast v_store void_ptr_t "store_to_i8" builder in

  ignore(L.build_store curr_vp curr_vpp builder);
```

```
    obs
  in


  (*
    A new custom type converter function is created for every subscription.
    Takes 4 void pointers, and internally converts them to correct types and
pushes the result
  *)
  let build_type_converter ((rtype: A.typ), typ1, typ2) (cls: observable_class) =
    let llrtype = match rtype with
      | A.Void -> i8_t
      | _ as x -> ltype_of_typ x
    in
    let lltyp1  = ltype_of_typ typ1 in
    let lltyp2  = ltype_of_typ typ2 in

    let nftype = L.function_type void_t ([| void_ptr_t; void_ptr_t; void_ptr_t;
void_ptr_t |]) in
    let nfunction = L.define_function "converter" nftype the_module in

    let nbuilder = L.builder_at_end context (L.entry_block nfunction) in
    L.set_value_name "r"  (L.params nfunction).(0);
    L.set_value_name "p1" (L.params nfunction).(1);
    L.set_value_name "p2" (L.params nfunction).(2);
    L.set_value_name "f"  (L.params nfunction).(3);

    let _rv  = L.build_bitcast (L.params nfunction).(0) (L.pointer_type llrtype)
"i8_to_r"  nbuilder in
    let _p1v = L.build_bitcast (L.params nfunction).(1) (L.pointer_type lltyp1)
"i8_to_p1" nbuilder in

    let build_call = match cls with
      | SingleObservable ->
        let dest_func_type = L.pointer_type (L.function_type llrtype ([| lltyp1
|])) in
        let _fv  = L.build_bitcast (L.params nfunction).(3) dest_func_type
"i8_to_f"  nbuilder in
```

```
        let p1 = L.build_load _p1v "p1" nbuilder in

        L.build_call _fv [| p1 |]
      | DoubleObservable ->
        let dest_func_type = L.pointer_type (L.function_type llrtype ([| lltyp1;
lltyp2 |])) in
        let _p2v = L.build_bitcast (L.params nfunction).(2) (L.pointer_type
lltyp2) "i8_to_p2" nbuilder in
        let _fv  = L.build_bitcast (L.params nfunction).(3) dest_func_type
"i8_to_f"  nbuilder in

        let p1 = L.build_load _p1v "p1" nbuilder in
        let p2 = L.build_load _p2v "p2" nbuilder in
        L.build_call _fv [| p1; p2 |]
    in

    let _ = match rtype with
      | A.Void -> build_call "" nbuilder;
      | _ ->
        let result = build_call "result" nbuilder in
        L.build_store result _rv nbuilder
    in
    ignore(L.build_ret_void nbuilder) ;
    nfunction
  in


  let rec oexpr vars builder ((_, oe) : soexpr) = match oe with
      SOId s -> L.build_load (lookup vars s) s builder
    | SMap(e, oe) ->
      (*   Basically SSubscribe, but returns obs  *)
      (* 1 *)

      let (rt, p1t) = match fst e with
        | A.Func([x], r) -> (r, x)
        | _ -> raise  (Failure "Subscriber must be a function")
      in

      let func_p = expr global_vars builder e in
      let upstream = oexpr global_vars builder oe in
```

```
      let obs = make_observable (rt, None) builder in

      let obs_store = L.define_global "sub_obs" (L.const_null obv_pt) the_module
in
      ignore(L.build_store obs obs_store builder) ;


      let obs_func_vpp = L.build_struct_gep obs 3 "__func" builder in
      let func_vp = L.build_bitcast func_p void_ptr_t "func_to_i8" builder in
      ignore(L.build_store func_vp obs_func_vpp builder);

      (* 2 *)
      ignore(L.build_call
          (lookup global_vars "subscribe") [| upstream; obs |] "" builder) ;

      let upstream_curr_vpp = L.build_struct_gep upstream 0 "__ups_curr_vpp"
builder in
      let upstream_curr_vp = L.build_load upstream_curr_vpp "__ups_curr_vp"
builder in
      let obs_upv_vpp = L.build_struct_gep obs 1 "__dwn_upv_vpp" builder in
      ignore(L.build_store upstream_curr_vp obs_upv_vpp builder) ;

      (* 3 *)
      let converter = build_type_converter (rt, p1t, A.Void) SingleObservable in
      let obs_converter_p = L.build_struct_gep obs 6 "__converter" builder in
      ignore(L.build_store converter obs_converter_p builder) ;


      (* 4 *)
      let _r  = L.build_load (L.build_struct_gep obs 0 "_r"  builder) "_r"
builder in
      let _p1 = L.build_load (L.build_struct_gep obs 1 "_p1" builder) "_p1"
builder in
      let _f  = L.build_load (L.build_struct_gep obs 3 "_f"  builder) "_f"
builder in
      let _c  = L.build_load (L.build_struct_gep obs 6 "_c"  builder) "_c"
builder in
      ignore(L.build_call _c [| _r; _p1; _p1; _f |] "" builder) ;
```

```
      (* 3 *)

      (* won't need but won't hurt to have *)
      ignore(L.build_call (lookup global_vars "onNext") [| obs |] "" builder) ;
      ; obs
    | SCombine(e, oe1, oe2) ->
      (*   Basically SSubscribe, but returns obs  *)
      (* 1 *)

      let (rt, p1t, p2t) = match fst e with
        | A.Func([x; y], r) -> (r, x, y)
        | _ -> raise  (Failure "Subscriber must be a function")
      in

      let func_p = expr global_vars builder e in
      let upstream1 = oexpr global_vars builder oe1 in
      let upstream2 = oexpr global_vars builder oe2 in
      let obs = make_observable (rt, None) builder in

      let obs_store = L.define_global "sub_obs" (L.const_null obv_pt) the_module
in
      ignore(L.build_store obs obs_store builder) ;


      let obs_func_vpp = L.build_struct_gep obs 3 "__func" builder in
      let func_vp = L.build_bitcast func_p void_ptr_t "func_to_i8" builder in
      ignore(L.build_store func_vp obs_func_vpp builder);

      (* 2 *)
      ignore(L.build_call
          (lookup global_vars "subscribe") [| upstream1; obs |] "" builder) ;
      ignore(L.build_call
          (lookup global_vars "subscribe") [| upstream2; obs |] "" builder) ;

      let upstream1_curr_vpp = L.build_struct_gep upstream1 0 "__ups1_curr_vpp"
builder in
      let upstream1_curr_vp = L.build_load upstream1_curr_vpp "__ups1_curr_vp"
builder in
      let obs_upv1_vpp = L.build_struct_gep obs 1 "__dwn_upv1_vpp" builder in
      ignore(L.build_store upstream1_curr_vp obs_upv1_vpp builder) ;
```

```
      let upstream2_curr_vpp = L.build_struct_gep upstream2 0 "__ups2_curr_vpp"
builder in
      let upstream2_curr_vp = L.build_load upstream2_curr_vpp "__ups2_curr_vp"
builder in
      let obs_upv2_vpp = L.build_struct_gep obs 2 "__dwn_upv2_vpp" builder in
      ignore(L.build_store upstream2_curr_vp obs_upv2_vpp builder) ;


      (* 3 *)
      let converter = build_type_converter (rt, p1t, p2t) DoubleObservable in
      let obs_converter_p = L.build_struct_gep obs 6 "__converter" builder in
      ignore(L.build_store converter obs_converter_p builder) ;


      (* 4 *)
      let _r  = L.build_load (L.build_struct_gep obs 0 "_r"  builder) "_r"
builder in
      let _p1 = L.build_load (L.build_struct_gep obs 1 "_p1" builder) "_p1"
builder in
      let _p2 = L.build_load (L.build_struct_gep obs 2 "_p2" builder) "_p2"
builder in
      let _f  = L.build_load (L.build_struct_gep obs 3 "_f"  builder) "_f"
builder in
      let _c  = L.build_load (L.build_struct_gep obs 6 "_c"  builder) "_c"
builder in
      ignore(L.build_call _c [| _r; _p1; _p2; _f |] "" builder ) ;


      (* 3 *)

      (* won't need but won't hurt to have *)
      ignore(L.build_call (lookup global_vars "onNext") [| obs |] "" builder) ;
      ; obs

    | SOBinop1(_, _, _) -> raise (Failure ("Not Implemented 2020"))
    | SOBinop2(_, _, _) -> raise (Failure ("Not Implemented 2020"))
    | SOBinop3(_, _, _) -> raise (Failure ("Not Implemented 2020"))
    | SOUnop(_, _) -> raise (Failure ("Not Implemented 2020"))
  in
```

```
  let build_global_stmt builder = function
      SExpr e -> ignore(expr global_vars builder e); builder
    | SDecl(t, s, e) ->
        let _ = match StringHash.find_opt global_vars s with
          | Some(_) -> raise (Failure "Cannot declare global more than once!")
          | None -> ()
        in

        let init t' = match t' with
            A.Float -> L.const_float float_t 0.0
          | A.Int   -> L.const_int i32_t 0
          | A.Bool  -> L.const_int i1_t 0
          | A.Char  -> L.const_int i8_t 0
          | A.Arr(_) -> L.const_null (L.pointer_type void_ptr_t)
          | A.Func(_, _) as f -> L.const_null (ltype_of_typ f)
          | A.Struct(_) as s -> L.const_pointer_null (ltype_of_typ s)
          | _ as x -> raise (Failure ("Cannot have global of type: " ^
A.string_of_typ x))
        in
        let store = L.define_global s (init t) the_module in
        StringHash.add global_vars s store;
        let e' = expr global_vars builder e in

        let _ = match t with
          | A.Arr(x) -> StringHash.add arr_types s x
          | _ -> ()
        in
        ignore(L.build_store e' store builder);
        builder
    | SStr_Def(s, b_list) ->
        let tlist = List.map fst b_list in
        let flist = List.map snd b_list in
        let ty = struct_t (list_to_lltype tlist)  in
        StringHash.add global_structs ("struct " ^ s) (ty, tlist, flist) ;
builder
    | _ -> raise (Failure "Global statement not implemented")
  in
```

```
  let build_obs_stmt builder = function
      SObs(_, _) -> builder

    | SODecl(lt, s, e) ->
      let e' = expr global_vars builder e in
      let lt' = match lt with
        | A.Observable x -> x
        | _ -> raise (Failure "Observable declaration must use observable")
      in
      let obv_ptr = make_observable (lt', Some e') builder in
      let store = L.define_global s (L.const_null (L.pointer_type obv_t))
the_module in
      ignore(L.build_store obv_ptr store builder) ;
      StringHash.add global_vars s store;
      builder
    | SOODecl(_, s, oe) ->
      let oe' = oexpr global_vars builder oe in
      let store = L.define_global s (L.const_null (L.pointer_type obv_t))
the_module in
      ignore(L.build_store oe' store builder) ;
      StringHash.add global_vars s store;
      builder
    | SOAssign(lt, s, e) ->
      let e' = expr global_vars builder e in
      let lt' = match lt with
        | A.Observable x -> x
        | _ -> raise (Failure "Observable declaration must use observable")
      in
      let obs = L.build_load (lookup global_vars s) s builder in

      let curr_vpp = L.build_struct_gep obs 0 "__curr_vpp" builder in
(* curr_vpp: i8**  *)
      let curr_vp = L.build_load curr_vpp "__curr_vp" builder in
(* curr_vp : i8*   *)
      let curr_p = L.build_bitcast curr_vp (L.pointer_type (ltype_of_typ lt'))
"i8_to_curr" builder in  (* curr_p  : i32*  *)
      ignore(L.build_store e' curr_p builder) ;
      ignore(L.build_call (lookup global_vars "onNext") [| obs |] "" builder) ;
      builder
```

```
    | SOOAssign(_, _, _) ->
      (* Maybe we should not allow this *)
      builder
    | SComplete(_, oe) ->
      let upstream = oexpr global_vars builder oe in
      ignore(L.build_call (lookup global_vars "complete") [| upstream |] ""
builder) ;
      builder
    | SSubscribe(_, e, oe) ->
      (*
        1. Create new observable
          1.1. allocate value holder in the stack
          1.2. malloc observable in the heap
          1.3. store function pointer to observable
          1.4. store 1.1 to observable
        2. Connect to upstream
          2.1. go to upstream and store itself as the child
          2.2. copy upstreamValue address to itself
          2.3.
        3. Define converter function
        4. Call function on upstream value
      *)

      (* 1 *)
      let (rt, p1t) = match fst e with
      | A.Func([x], r) -> (r, x)
      | _ -> raise  (Failure "Subscriber must be a function")
      in
      let func_p = expr global_vars builder e in

      let upstream = oexpr global_vars builder oe in
      let obs = make_observable (rt, None) builder in

      let obs_store = L.define_global "sub_obs" (L.const_null obv_pt) the_module
in
      ignore(L.build_store obs obs_store builder) ;


      let obs_func_vpp = L.build_struct_gep obs 3 "__func" builder in
```

```
      let func_vp = L.build_bitcast func_p void_ptr_t "func_to_i8" builder in
      ignore(L.build_store func_vp obs_func_vpp builder);

      (* 2 *)

      let upstream_curr_vpp = L.build_struct_gep upstream 0 "__ups_curr_vpp"
builder in
      let upstream_curr_vp = L.build_load upstream_curr_vpp "__ups_curr_vp"
builder in
      let obs_upv_vpp = L.build_struct_gep obs 1 "__dwn_upv_vpp" builder in
      ignore(L.build_store upstream_curr_vp obs_upv_vpp builder) ;
      ignore(L.build_call
          (lookup global_vars "subscribe") [| upstream; obs |] "" builder) ;

      (* 3 *)
      let converter = build_type_converter (rt, p1t, A.Void) SingleObservable in
      let obs_converter_p = L.build_struct_gep obs 6 "__converter" builder in
      ignore(L.build_store converter obs_converter_p builder) ;


      (* 4 *)
      let _r  = L.build_load (L.build_struct_gep obs 0 "_r"  builder) "_r"
builder in
      let _p1 = L.build_load (L.build_struct_gep obs 1 "_p1" builder) "_p1"
builder in
      let _f  = L.build_load (L.build_struct_gep obs 3 "_f"  builder) "_f"
builder in
      let _c  = L.build_load (L.build_struct_gep obs 6 "_c"  builder) "_c"
builder in
      ignore(L.build_call _c [| _r; _p1; _p1; _f |] "" builder) ;



      (* 4 *)

      (* won't need but won't hurt to have *)
      ignore(L.build_call (lookup global_vars "onNext") [| obs |] "" builder) ;
      builder
    | _ -> raise (Failure ("Not Implemented 2100"))
  in
```

```
  let translate_line glob = match glob with
        SStmt stmt -> ignore(build_global_stmt global_builder stmt)
      | SObs_Stmt obs_stmt -> ignore(build_obs_stmt global_builder obs_stmt)

  in
  let () = List.iter translate_line globs in
  let () = add_terminal global_builder (L.build_ret (L.const_int i32_t 0)) in

  (* define_subscribe *)
  let _ =
    (*
    void subscribe(observable* upstream, observable* downstream) {

      subscription *new = (subscription^) malloc(sizeof(subscription));
      new->__obs = downstream;
      new->__next = upstream->subscription;

      upstream->subscription = new
    }
    *)

    let subscribe_function = StringHash.find global_vars "subscribe" in
    let builder = L.builder_at_end context (L.entry_block subscribe_function) in
    L.set_value_name "upstream" (L.params subscribe_function).(0);
    L.set_value_name "downstream" (L.params subscribe_function).(1);
    let up_local   = L.build_alloca obv_pt "upstream" builder in
    let down_local = L.build_alloca obv_pt "downstream" builder in
    ignore(L.build_store (L.params subscribe_function).(0) up_local builder) ;
    ignore(L.build_store (L.params subscribe_function).(1) down_local builder) ;
    let upstream = L.build_load up_local "__ups" builder in      (* upstream   :
observable*  *)
    let downstream = L.build_load down_local "__dws" builder in  (* downstream :
observable*  *)


    let next' = L.build_struct_gep upstream 4 "__subscription_pp" builder in  (*
next' : subscription**  &(upstream->sub) *)
```

```
    let next  = L.build_load next' "__subscription_p" builder in              (*
next  : subscription*   upstream->sub    *)


    let new_sub = L.build_malloc subscription_t "new" builder in     (* new_sub :
subscription*                    *)
    let n_next' = L.build_struct_gep new_sub 0 "__next" builder in   (* n_next' :
subscription**  &(new_sub->next) *)
    let n_obs'  = L.build_struct_gep new_sub 1 "__obs" builder in    (* n_obs'  :
observable**    &(new_sub->obs)  *)
    ignore(L.build_store next n_next' builder) ;
    ignore(L.build_store downstream n_obs' builder) ;

    ignore(L.build_store new_sub next' builder) ;
    L.build_ret_void builder;
  in

  (* define_complete *)
  let _ =
    (*
    void complete(observable* upstream) {
      upstream->__next = Null
    }
    *)
    let complete_function = StringHash.find global_vars "complete" in
    let builder = L.builder_at_end context (L.entry_block complete_function) in
    L.set_value_name "upstream" (L.params complete_function).(0);
    let up_local = L.build_alloca obv_pt "upstream" builder in
    ignore(L.build_store (L.params complete_function).(0) up_local builder) ;
    let upstream = L.build_load up_local "__ups" builder in
    let next' = L.build_struct_gep upstream 4 "__subscription_pp" builder in  (*
next' : subscription**  &(upstream->sub) *)
    let new_sub = L.const_pointer_null subscription_pt in     (* new_sub :
subscription*                    *)
    ignore(L.build_store new_sub next' builder) ;
    L.build_ret_void builder;

  in

  (* define_next *)
```

```
  let _ =
    (*
    void onNext(observable* obs) {

      subscription* current = obs->__currentscription;
      while (current != NULL) {
        observable* d = current->__obs;
        d->__curr = d->__func(d->__upstreamValue);
        onNext(d);
        current = current->__next;
        return;
      }
      return;
    }
    *)

    (* Define function *)
    let next_function = StringHash.find global_vars "onNext" in
    let builder = L.builder_at_end context (L.entry_block next_function) in
    L.set_value_name "upstream" (L.params next_function).(0);
    let up_local = L.build_alloca obv_pt "upstream" builder in
    ignore(L.build_store (L.params next_function).(0) up_local builder) ;
    let upstream = L.build_load up_local "__ups" builder in


    let _sub' = L.build_struct_gep upstream 4 "__sub_pp" builder in      (* sub'
: subscription**  *)
    let _sub  = L.build_load _sub' "__sub_p" builder in                  (* sub
: subscription*   *)
    let current' = L.build_alloca subscription_pt "current_p" builder in (*
current' : subscription**  *)
    ignore(L.build_store _sub current' builder) ;

    (* child blocks *)
    let while_pred_bb = L.append_block context "while_pred" next_function in
    let while_pred_builder = L.builder_at_end context while_pred_bb in

    let while_body_bb = L.append_block context "while_body" next_function in
    let while_body_builder = L.builder_at_end context while_body_bb in
```

```
    let merge_bb = L.append_block context "merge" next_function in
    let merge_builder = L.builder_at_end context merge_bb in


    (* while_pred block *)
    let current = L.build_load current' "current" while_pred_builder in    (*
current : subscription*   *)

    let cond = L.build_icmp L.Icmp.Ne (L.const_pointer_null subscription_pt)
current "tmp" while_pred_builder in

    let d' = L.build_struct_gep current 1 "__d_ref" while_body_builder in  (* d'
: observable**  *)
    let d  = L.build_load d' "__d" while_body_builder in                   (* d
: observable*   *)

    let next_sub' = L.build_struct_gep current 0 "__next_p" while_body_builder in
(* next_sub' : subscription**  *)
    let next_sub  = L.build_load next_sub' "__next" while_body_builder in
(* next_sub  : subscription*   *)
    ignore(L.build_store next_sub current' while_body_builder) ;



    (* single block *)

    let _r  = L.build_load (L.build_struct_gep d 0 "_r"  while_body_builder) "_r"
while_body_builder in
    let _p1 = L.build_load (L.build_struct_gep d 1 "_p1" while_body_builder)
"_p1" while_body_builder in
    let _p2 = L.build_load (L.build_struct_gep d 2 "_p2" while_body_builder)
"_p2" while_body_builder in
    let _f  = L.build_load (L.build_struct_gep d 3 "_f"  while_body_builder) "_f"
while_body_builder in
    let _c  = L.build_load (L.build_struct_gep d 6 "_c"  while_body_builder) "_c"
while_body_builder in
    let _ = L.build_call _c [| _r; _p1; _p2; _f |] "" while_body_builder in

    (*    onNext(d); *)
    ignore(L.build_call next_function [| d |] "" while_body_builder) ;
```

```
    (* Connect blocks *)
    ignore(L.build_br while_pred_bb builder) ;
    ignore(L.build_cond_br cond while_body_bb merge_bb while_pred_builder) ;
    ignore(L.build_br while_pred_bb while_body_builder);
    ignore(L.build_ret_void merge_builder)
  in

  the_module
```

8.1.1.6 ./sast.ml

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

open Ast

type sexpr = typ * sx
and sx =
    SLiteral of int
  | SBliteral of bool
  | SFliteral of string
  | SChliteral of char
  | SAliteral of typ * sexpr list
  | SId of string
  | SSid of string
  | SBinop of sexpr * op * sexpr
  | SUnop of uop * sexpr
  | SCall of sexpr * sexpr list
  | SBCall of string * sexpr list (* Built-in function call *)
  | SRef of string * string * string
  | SArr_Ref of sexpr * sexpr
  | SFuncExpr of bind list * typ * sstmt list
  | SSliteral of sexpr list
  | SLen of sexpr
  | SNoexpr
```

```
  | SNull

and

sstmt =
    SBlock of sstmt list
  (* | Obs of string *)
  | SExpr of sexpr
  | SIf of typ * string * sexpr * sexpr * sexpr
  | SReturn of sexpr
  | SPrint of sexpr
  | SDecl of typ * string * sexpr
  | SStr_Def of string * bind list

type soexpr = typ * sox
and sox =
    SOId of string
  | SOBinop1 of soexpr * op * sexpr   (* For map operation *)
  | SOBinop2 of sexpr * op * soexpr
  | SOBinop3 of soexpr * op * soexpr  (* For combine operation *)
  | SOUnop of uop * soexpr
  | SMap of sexpr * soexpr
  | SCombine of sexpr * soexpr * soexpr

type sobs_stmt =
    SObs of typ * string
  | SOExpr of soexpr
  | SODecl of typ * string * sexpr
  | SOODecl of typ * string * soexpr
  | SOAssign of typ * string * sexpr
  | SOOAssign of typ * string * soexpr
  | SOArr_Decl of typ * string * sexpr list
  | SOStr_Decl of typ * string * sexpr list
  | SSubscribe of string * sexpr * soexpr
  | SComplete of string * soexpr

type sglob =
    SStmt of sstmt
  | SObs_Stmt of sobs_stmt
```

```
type sprogram = sglob list

(* Pretty-printing functions *)

let rec string_of_sexpr (t, e) =
  "(" ^ string_of_typ t ^ " : " ^ (match e with
    SLiteral(l) -> string_of_int l
  | SBliteral(b) -> if b then "TRUE" else "FALSE"
  | SFliteral(l) -> l
  | SChliteral(l) -> "'" ^ String.make 1 l ^ "'"
  | SAliteral(_, l) -> "[" ^ String.concat "," (List.map string_of_sexpr l) ^ "]"
  | SId(s) -> "(id: " ^ s ^ ")"
  | SSid(s) -> "(struct: " ^ 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
  | SCall(ef, el) ->
      string_of_sexpr ef ^ "(" ^ String.concat ", " (List.map string_of_sexpr el)
^ ")"
  | SBCall(s, el) -> (* Built-in function call *)
      s ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^ ")"
  | SArr_Ref(e1, e2) -> string_of_sexpr e1 ^ "[" ^ string_of_sexpr e2 ^ "]"
  | SRef(s1, _, s2) -> s1 ^ "." ^ s2
  | SFuncExpr(bind_list, _, stmt_list) -> "(" ^
      String.concat ", " (List.map string_of_bind bind_list) ^ ") -> {" ^
      String.concat "" (List.map string_of_sstmt stmt_list) ^ "}"
  | SSliteral(e_list) -> "(sliteral: { " ^ String.concat ", " (List.map
string_of_sexpr e_list) ^ " })"
  | SLen(e) -> string_of_sexpr e ^ ".length"
  | SNull -> "null"
  | SNoexpr -> ""
       ) ^ ")"

and

string_of_sstmt = function
    SBlock(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_sstmt stmts) ^ "}\n"
  | SExpr(sexpr) -> string_of_sexpr sexpr ^ ";\n";
  | SReturn(sexpr) -> "return " ^ string_of_sexpr sexpr ^ ";\n";
```

```
  | SPrint(expr) -> "print(" ^ string_of_sexpr expr ^ ");\n"
  | SDecl(t, s, expr) -> string_of_typ t ^ " " ^ s ^ " = "^ string_of_sexpr expr
^ ";\n"
  | SStr_Def(s, bind_list) -> "struct " ^ s ^ " { " ^
      String.concat "\n" (List.map string_of_bind bind_list) ^ "\n};\n"
  | SIf(t, s, e1, e2, e3) -> string_of_typ t ^ " " ^ s ^ " = if(" ^
string_of_sexpr e1 ^ ") "
    ^ string_of_sexpr e2 ^ " else " ^ string_of_sexpr e3


let rec string_of_soexpr (t, e) =
  "(" ^ string_of_typ t ^ " : " ^ (match e with
    SOId(s) -> s
  | SOBinop1(e1, o, e2) ->
    string_of_soexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_sexpr e2
  | SOBinop2(e1, o, e2) ->
    string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_soexpr e2
  | SOBinop3(e1, o, e2) ->
    string_of_soexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_soexpr e2
  | SOUnop(o, e) -> string_of_uop o ^ string_of_soexpr e
  | SMap(e, oe) -> "map(" ^ string_of_sexpr e ^ ", " ^ string_of_soexpr oe ^ ")"
  | SCombine(e, oe1, oe2) -> "combine(" ^ string_of_sexpr e ^ ", " ^
      string_of_soexpr oe1 ^ ", " ^ string_of_soexpr oe2 ^ ")"

  ) ^ ")"

let string_of_sobs_stmt = function
    SObs(t, s) -> string_of_typ t ^ " " ^ s ^ ";\n"
  | SOExpr(e) -> string_of_soexpr e ^ ";\n"
  | SODecl(t, s, e) -> string_of_typ t ^ " " ^ s ^ " = " ^ string_of_sexpr e ^
";\n"
  | SOODecl(t, s, e) -> string_of_typ t ^ " " ^ s ^ " = " ^ string_of_soexpr e ^
";\n"
  | SOAssign(_, s, e) -> s ^ " = " ^ string_of_sexpr e ^ ";\n"
  | SOOAssign(_, s, e) -> s ^ " = " ^ string_of_soexpr e ^ ";\n"
  | SOArr_Decl(t, s, expr_list) -> string_of_typ t ^ " " ^ s ^ " = [" ^
      String.concat ", " (List.map string_of_sexpr expr_list) ^ "];\n"
  | SOStr_Decl(t, s, expr_list) -> string_of_typ t ^ " " ^ s ^ " = {" ^
      String.concat ", " (List.map string_of_sexpr expr_list) ^ "};\n"
  | SSubscribe(s, e, oe) ->
```

```
      s ^ "(" ^ string_of_sexpr e ^ ", " ^ string_of_soexpr oe ^ ");\n"
  | SComplete(s, oe) ->
      s ^ "(" ^ string_of_soexpr oe ^ ");\n"

let sstr_of_glob = function
   SStmt(stmt) -> string_of_sstmt stmt
  | SObs_Stmt(obs_stmt) -> string_of_sobs_stmt obs_stmt

let sstring_of_program (globs) =
  String.concat "" (List.map sstr_of_glob globs) ^ "\n"
```

8.1.1.7 ./scanner.mll

```
(* Ocamllex scanner for Seaflow *)

{ open Seaflowparse }

let digit = ['0' - '9']
let char = ['a'-'z'] | ['A'-'Z'] | ['0'-'9'] | ' ' | '!' | ['#'-'~']
let charlit = "\'"char"\'"
let strlit = "\""char*"\""
let digits = digit+

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"     { comment lexbuf }           (* Comments *)
| '('      { LPAREN }
| ')'      { RPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
| '['      { LBRAKT }
| ']'      { RBRAKT }
| "->"     { ARROW }
| '.'      { DOT }
| ';'      { SEMI }
| ','      { COMMA }
| '+'      { PLUS }
| '-'      { MINUS }
```

```
| '*'      { TIMES }
| '/'      { DIVIDE }
| '='      { ASSIGN }
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
| ">"      { GT }
| ">="     { GEQ }
| "&&"     { AND }
| "||"     { OR }
| "if"     { IF }
| "else"   { ELSE }
| "return" { RETURN }
| "int"    { INT }
| "float"  { FLOAT }
| "char"   { CHAR }
| "void"   { VOID }
| "struct" { STRUCT }
| "map"       { MAP }
| "combine"   { COMBINE }
| "subscribe" { SUBSCRIBE }
| "complete"  { COMPLETE }
| charlit as lxm { CHLIT(lxm.[1])}
| strlit as lxm { STRLIT (String.escaped lxm)}
| digits as lxm { LITERAL(int_of_string lxm) }
| digits '.'  digit* ( ['e' 'E'] ['+' '-']? digits )? as lxm { FLIT(lxm) }
| ['a'-'z']['a'-'z' 'A'-'Z' '0'-'9' '_']*            as lxm { ID(lxm) }
| ['A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*            as lxm { SID(lxm) }
| '$'['a'-'z']['a'-'z' 'A'-'Z' '0'-'9' '_']*  as lxm { OBS(lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

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

8.1.1.8 ./seaflow.ml

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

type action = Ast | Sast | LLVM_IR | Compile

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

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

8.1.1.9 ./seaflowparse.mly

```
/* Ocamlyacc parser for Seaflow */
```

```
%{
open Ast
%}

%token MAP COMBINE SUBSCRIBE COMPLETE
%token SEMI LPAREN RPAREN LBRACE RBRACE LBRAKT RBRAKT COMMA PLUS MINUS TIMES
DIVIDE ASSIGN ARROW DOT
%token EQ NEQ LT LEQ GT GEQ AND OR
%token RETURN IF ELSE INT FLOAT VOID CHAR STRUCT NULL
%token <int> LITERAL
%token <string> ID FLIT OBS SID STRLIT
%token <char> CHLIT
%token EOF


%start program
%type <Ast.program> program

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE
%left LBRAKT
%left DOT

%%

program:
  glob_lines EOF { List.rev $1 }

glob_lines:
     { [] }
  | glob_lines glob_line { $2 :: $1 }
```

```
glob_line:
  | fdecl    { Fdecl($1) }
  | stmt     { Stmt($1) }
  | obs_stmt { Obs_Stmt($1) }

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

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

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

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

typ:
    INT     { Int   }
  | FLOAT   { Float }
  | VOID    { Void  }
  | CHAR    { Char }
  | typ LBRAKT RBRAKT { Arr($1) }             /* array */
  | STRUCT SID        { Struct($2) }          /* struct */
  | LPAREN typ_list RPAREN ARROW LPAREN typ RPAREN { Func(List.rev $2, $6) } /*
for higher order function */

typ_list:
    typ                { [$1] }
  | typ_list COMMA typ { $3 :: $1 }

sdecl_list:
    typ ID SEMI                { [($1,$2)] }
```

```
  | sdecl_list typ ID SEMI            { ($2,$3) :: $1 }

stmt:
    expr SEMI                               { Expr $1               }
  | RETURN expr_opt SEMI                    { Return $2             }
  | typ ID ASSIGN expr SEMI                  { Decl($1, $2, $4) }
  | STRUCT SID LBRACE sdecl_list RBRACE SEMI { Str_Def($2, List.rev $4) }
  | typ ID ASSIGN IF LPAREN expr RPAREN expr ELSE expr SEMI   { If($1, $2, $6,
$8, $10) }


obs_stmt:
    obs_expr SEMI                            { OExpr $1 }
  | OBS ASSIGN expr SEMI                     { OAssign($1, $3)         }
  | OBS ASSIGN obs_expr SEMI                 { OOAssign($1, $3) }
  | typ OBS SEMI                             { Obs(Observable($1), $2) }
  | typ OBS ASSIGN expr SEMI                 { ODecl(Observable($1), $2, $4) }
  | typ OBS ASSIGN obs_expr SEMI             { OODecl(Observable($1), $2, $4) }

  | SUBSCRIBE LPAREN expr COMMA obs_expr RPAREN SEMI { Subscribe($3, $5) }
  | COMPLETE  LPAREN obs_expr RPAREN SEMI            { Complete($3) }


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

expr:
    LITERAL          { Literal($1)            }
  | NULL             { Null                   }
  | FLIT             { Fliteral($1)           }
  | CHLIT            { Chliteral($1)          }
  | STRLIT           { Strliteral($1)         }
  | ID               { Id($1)                 }
  | SID              { Sid($1)                }
  | expr DOT ID      { Ref($1, $3)            }
  | expr PLUS   expr { Binop($1, Add,   $3)   }
  | expr DIVIDE expr { Binop($1, Div,   $3)   }
  | expr MINUS  expr { Binop($1, Sub, $3)     }
  | expr TIMES  expr { Binop($1, Mult, $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)   }
  | LBRAKT args_list RBRAKT { Aliteral(List.rev $2)         }

  | MINUS expr       { Unop(Neg, $2)          }
  | LBRACE args_list RBRACE                   { Sliteral(List.rev $2)}
  | expr LBRAKT expr RBRAKT                     { Arr_Ref($1, $3)  }
  | ID LPAREN args_opt RPAREN                 { Call(Id($1), $3) }
  | LPAREN expr RPAREN LPAREN args_opt RPAREN { Call($2, $5)     }
  | LPAREN expr RPAREN { $2                   }
  | LPAREN formals_opt RPAREN ARROW LBRACE func_body RBRACE
      { FuncExpr(List.rev $2, List.rev $6) }   /* anonymous function */

obs_expr:
  | obs_var              { OId($1) }
  | obs_expr PLUS   expr { OBinop1($1, Add,   $3)   }
  | obs_expr DIVIDE expr { OBinop1($1, Div,   $3)   }
  | obs_expr MINUS  expr { OBinop1($1, Sub, $3)     }
  | obs_expr TIMES  expr { OBinop1($1, Mult, $3)    }
  | obs_expr EQ     expr { OBinop1($1, Equal, $3)   }
  | obs_expr NEQ    expr { OBinop1($1, Neq,   $3)   }
  | obs_expr LT     expr { OBinop1($1, Less,  $3)   }
  | obs_expr LEQ    expr { OBinop1($1, Leq,   $3)   }
  | obs_expr GT     expr { OBinop1($1, Greater, $3) }
  | obs_expr GEQ    expr { OBinop1($1, Geq,   $3)   }
  | obs_expr AND    expr { OBinop1($1, And,   $3)   }
  | obs_expr OR     expr { OBinop1($1, Or,    $3)   }

  | expr PLUS   obs_expr { OBinop2($1, Add,   $3)   }
  | expr DIVIDE obs_expr { OBinop2($1, Div,   $3)   }
  | expr MINUS  obs_expr { OBinop2($1, Sub, $3)     }
  | expr TIMES  obs_expr { OBinop2($1, Mult, $3)    }
  | expr EQ     obs_expr { OBinop2($1, Equal, $3)   }
  | expr NEQ    obs_expr { OBinop2($1, Neq,   $3)   }
```

```
  | expr LT     obs_expr { OBinop2($1, Less,  $3)   }
  | expr LEQ    obs_expr { OBinop2($1, Leq,   $3)   }
  | expr GT     obs_expr { OBinop2($1, Greater, $3) }
  | expr GEQ    obs_expr { OBinop2($1, Geq,   $3)   }
  | expr AND    obs_expr { OBinop2($1, And,   $3)   }
  | expr OR     obs_expr { OBinop2($1, Or,    $3)   }

  | obs_expr PLUS   obs_expr { OBinop3($1, Add,   $3)   }
  | obs_expr DIVIDE obs_expr { OBinop3($1, Div,   $3)   }
  | obs_expr MINUS  obs_expr { OBinop3($1, Sub, $3)     }
  | obs_expr TIMES  obs_expr { OBinop3($1, Mult, $3)    }
  | obs_expr EQ     obs_expr { OBinop3($1, Equal, $3)   }
  | obs_expr NEQ    obs_expr { OBinop3($1, Neq,   $3)   }
  | obs_expr LT     obs_expr { OBinop3($1, Less,  $3)   }
  | obs_expr LEQ    obs_expr { OBinop3($1, Leq,   $3)   }
  | obs_expr GT     obs_expr { OBinop3($1, Greater, $3) }
  | obs_expr GEQ    obs_expr { OBinop3($1, Geq,   $3)   }
  | obs_expr AND    obs_expr { OBinop3($1, And,   $3)   }
  | obs_expr OR     obs_expr { OBinop3($1, Or,    $3)   }

  | MAP LPAREN expr COMMA obs_expr RPAREN { Map($3, $5) }
  | COMBINE LPAREN expr COMMA obs_expr COMMA obs_expr RPAREN { Combine($3, $5,
$7) }


  | MINUS obs_expr         { OUnop(Neg, $2)       }
  | LPAREN obs_expr RPAREN { $2                   }

obs_var:
    OBS             { $1 }

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

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

8.1.1.10 ./semant.ml

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

open Ast
open Sast

module StringMap = Map.Make(String)

module StringHash = Hashtbl.Make(struct
  type t = string
  let equal x y = x = y
  let hash = Hashtbl.hash
end)

let global_vars = StringHash.create 10
(* let function_decls = StringHash.create 10 *)
let struct_defs = StringHash.create 10
let struct_by_body = StringHash.create 10

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

   Check each global variable, then check each function *)


let match_struct_element_name name (_, element) =
  if name = element then true else false


(* Observable can't be observable of observable *)
let get_observable_inner_type (ot: typ) : typ = match ot with
  | Observable(t) -> (match t with
    | Observable(_)
    | Func(_, _) -> raise (Failure ("Observable can't be observable of observable
or func"))
    | _ as x -> x)
  | _ as x -> raise (Failure (string_of_typ x ^ " is not an observable"))
```

```
let check (globs) =

  let _ = StringHash.add global_vars "printi" (Func([Int], Int)) in
  let _ = StringHash.add global_vars "printf" (Func([Float], Int)) in
  let _ = StringHash.add global_vars "printc" (Func([Char], Int)) in
  let _ = StringHash.add global_vars "prints" (Func([Arr(Char)], Void)) in

  let type_of_identifier vars s =
    try StringHash.find vars s
    with Not_found -> raise (Failure ("undeclared identifier " ^ s))
  in

  let find_body s = try StringHash.find struct_by_body (string_of_typ s)
              with Not_found -> raise (Failure ("struct type not found: " ^
string_of_typ s))
       in

  let check_assign lvaluet rvaluet err =
    match (lvaluet, rvaluet) with
      (Struct(_), Struct(_)) -> let xbody = find_body lvaluet in
        let ybody = find_body rvaluet in
        let same = xbody = ybody in
        if same then lvaluet else raise (Failure err)
    | (Struct(_), Sbody(_)) -> let xbody = find_body lvaluet in
        let same = xbody = rvaluet in
        if same then lvaluet else raise (Failure err)
    | _ -> let same = lvaluet = rvaluet in
        if same then lvaluet else raise (Failure err)
  in

  let rec expr vars = function
      Literal  l -> (Int, SLiteral l)
    | Bliteral l -> (Bool, SBliteral l)
    | Fliteral l -> (Float, SFliteral l)
    | Chliteral c -> (Char, SChliteral c)
    | Strliteral s ->
      let ty = Char in
      let split =
```

```
        let rec exp i l =
          if i < 2 then l else exp (i - 1) (Chliteral(s.[i]) :: l) in
        exp (String.length s - 3) []
      in
      let e = List.map (expr vars) split in
      (Arr(ty), SAliteral(ty, e))
    | Aliteral a ->
      let (ty, _) = expr vars (List.hd a) in
      let expr_list = List.map (expr vars) a in
      let compare x (y,_) =
        if x = y then () else raise (Failure("array literal: type mismatch " ^
string_of_typ x ^ " != " ^ string_of_typ y))
      in let _ = List.map (compare ty) expr_list in
      (Arr(ty), SAliteral (ty, expr_list))
    | Id s       -> (type_of_identifier vars s, SId s)
    | Ref(e, s) ->
      let (t', e') = expr vars e in
      (* this is how we allow reference of array literals *)
      let str_name = match e' with
        | SId(x) -> x
        | SAliteral(_,_) -> "NONE"
        | _ -> raise (Failure ("Struct field"))
      in
      let struct_ref ty str =
        let l = StringHash.find struct_defs (string_of_typ ty) in
        let element = List.find_opt (match_struct_element_name str) l in
        let element_type = match element with
          | Some (t2, _) -> t2
          | None -> raise (Failure ("field " ^ s ^ " is not part of this
struct"))
        in (element_type, SRef(str_name, string_of_typ ty, str))
      in
      let arr_len name str =
        let _ = if (str = "length") then () else raise (Failure ("invalid
reference: is not struct or array length")) in
        let ty_of_lit = function
            | SAliteral(x, _) -> Arr(x)
            | _ -> raise (Failure ("shouldn't happen"))
        in
```

```
        let ty = if (name = "NONE") then (ty_of_lit e') else type_of_identifier
vars name in
        let _ = try typ_of_arr ty
            with Match_failure(_) -> raise (Failure ("cannot take length of type
" ^ string_of_typ ty)) in
        let e' = expr vars e in
        (Int, SLen(e'))
      in
      let found = StringHash.mem struct_defs (string_of_typ t') in
      let ret = match found with
        false -> arr_len str_name s
        | true -> struct_ref t' s
      in ret
    | Sliteral(expr_list) ->
      let e' = List.map (expr vars) expr_list in
      let ty = List.map fst e' in
      (Sbody(ty), SSliteral(e'))
    | Arr_Ref(e1, e2) ->
      let ty = match (expr vars e1) with
        | (_, SId(s)) -> typ_of_arr (type_of_identifier vars s)
        | (_, SAliteral(t, _)) -> t
        | _ -> raise (Failure ("can only reference an array!"))
      in
      let e1' = expr vars e1 in
      (* basically just check that the array exists and that expr is an int *)
      let (idx_ty, e') = expr vars e2 in
      let _ = if idx_ty = Int then ()
          else raise(Failure ("array index must be of type int, not " ^
string_of_typ idx_ty)) in
      (ty, SArr_Ref(e1', (idx_ty, e')))
    | Noexpr     -> (Void, SNoexpr)
    | Binop(e1, op, e2) as e ->
        let (t1, e1') = expr vars e1
        and (t2, e2') = expr vars e2 in
        (* All binary operators require operands of the same type *)
        let same = t1 = t2 in
        (* Determine expression type based on operator and operand types *)

        (match op with
          | And | Or ->
```

```
            let (_t1, _e1') = match t1 with
              | Int -> expr vars (Binop(e1, Neq, Literal(0)))
              | Bool -> (t1, e1')
              | _ -> raise (Failure "&& and || must be used with integers")
            in
            let (_t2, _e2') = match t2 with
              | Int -> expr vars (Binop(e2, Neq, Literal(0)))
              | Bool -> (t2, e2')
              | _ -> raise (Failure "&& and || must be used with integers")
            in (Bool, SBinop((_t1, _e1'), op, (_t2, _e2')))
          | _ -> let ty = match op with
              Add | Sub | Mult | Div when same && t1 = Int   -> Int
            | Add | Sub | Mult | Div when same && t1 = Float -> Float
            | Add | Sub | Mult | Div when same && t1 = Char  -> Char
            | Equal | Neq            when same               -> Bool
            | Less | Leq | Greater | Geq
                     when same && (t1 = Int || t1 = Float || t1 = Char) -> Bool
            | Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater | Geq
when ((t1 = Float && t2 = Int) || (t1 = Int && t2 = Float)) -> Float
            | Add when (typ_of_arr t1) = (typ_of_arr t2) -> t1
            | _ -> raise (Failure ("illegal binary operator "  ^
                                   string_of_typ t1 ^ " " ^ string_of_op op ^ "
" ^
                                   string_of_typ t2 ^ " in " ^ string_of_expr
e))
            in (ty, SBinop((t1, e1'), op, (t2, e2'))))
    | Unop(op, e) ->
        let (ty, e') = expr vars e in (ty, SUnop(op, (ty, e')))
    | Call(f, args) as call ->
        let (t', f') = expr vars f in

        let (formals, rtype) = match t' with
          | Func(formals, rtype) -> (formals, rtype)
          | _ ->  raise (Failure ("must be Func type"))
        in

        let param_length = List.length formals in
        if List.length args != param_length then
          raise (Failure ("expecting " ^ string_of_int param_length ^
                          " arguments in " ^ string_of_expr call))
```

```
        else let check_call ft e =
          let (et, e') = expr vars e in
          let err = "illegal argument: found "  ^ string_of_typ et ^
              " expected "  ^ string_of_typ ft ^  " in "  ^ string_of_expr e
          in (check_assign ft et err, e')
        in
        let args' = List.map2 check_call formals args in
        (match f' with
          | SId(x) when x = "???" -> (rtype, SBCall(x, args'))
          | _ as x                -> (rtype, SCall((t', x), args')))
    | FuncExpr(params, stmts) ->
        let (ftype, rtype, sstmts) = check_func_decl vars (params, stmts) in
        (ftype, SFuncExpr(params, rtype, sstmts))
    | _ -> raise(Failure ("expression not implemented"))
  and check_stmt vars = function
      Expr e -> SExpr (expr vars e)
    | Decl(lt, var, e) as ex ->
      let _ = match lt with
          Void -> raise (Failure "cannot assign to void type!")
        | _ -> ()
      in
      let _ = match (StringHash.find_opt vars var) with
      | Some(_) -> raise (Failure "variable has already been assigned!")
      | None -> ()
      in
      StringHash.add vars var lt ;

      let (rt, e') = expr vars e in
      let err = "illegal assignment " ^ string_of_typ lt ^ " = " ^
        string_of_typ rt ^ " in " ^ string_of_stmt ex in
      ignore(check_assign lt rt err) ; SDecl(lt, var, (rt, e'))
    | Return e -> let (t, e') = expr vars e in SReturn (t, e')
    | Str_Def(s, b_list) ->
      let tlist = List.map fst b_list in
      let name = ("struct " ^ s) in
      StringHash.add struct_defs name b_list ; StringHash.add struct_by_body name
(Sbody(tlist)) ;
      SStr_Def(s, b_list)
    | Block sl ->
      let rec check_stmt_list v = function
```

```
          [Return _ as s] -> [check_stmt v s]
        | Return _ :: _   -> raise (Failure "nothing may follow a return")
        | Block sl :: ss  -> check_stmt_list v (sl @ ss) (* Flatten blocks *)
        | s :: ss         -> let parsed_stmt = check_stmt v s in parsed_stmt ::
check_stmt_list v ss
        | []              -> []
      in SBlock(check_stmt_list vars sl)
    | If(ltyp, var, cond, e1, e2) ->
      let (tc, c') = expr vars cond
      and (t1, e1') = expr vars e1
      and (t2, e2') = expr vars e2 in
      let same = t1 = t2 in
      (* e1 and e2 must be same type *)
      let rtyp = match t1 with
        Int when same -> Int
      | Float when same -> Float
      | Char when same -> Char
      | Arr(x) when same -> Arr(x)
      | _ -> raise (Failure ("illegal if; types must match in then and else
branches")) in
      let same2 = ltyp = rtyp in
      let _ = match ltyp with
          Int when same2 -> Int
        | Float when same2 -> Float
        | Char when same2 -> Char
        | Arr(x) when same2 -> Arr(x)
        | _ -> raise (Failure ("illegal if; types must match for lval and rval"))
in
      StringHash.add vars var ltyp ;
      SIf(ltyp, var, (tc, c'), (t1, e1'), (t2, e2'))
    | _ -> raise (Failure ("statement not implemented"))


  and check_func_decl vars anon_func : (Ast.typ * Ast.typ * Sast.sstmt list) =
    let (params, body) = anon_func in

    let local_vars = StringHash.copy vars in
    let add_to_hash (vtype, vname) =
      StringHash.add local_vars vname vtype
    in
```

```
  List.iter add_to_hash params;

  let sbody = match check_stmt local_vars (Block body) with
      SBlock(sl) -> sl
    | _ -> raise (Failure ("internal error2: block didn't become a block?"))
  in

  let check_param param = (match param with
      Int -> ()
    | Void -> raise (Failure ("cannot have void formal!"))
    | _ -> ()
  )
  in

  let param_types = List.map fst params in
  let _ = List.map check_param param_types in

  let check_empty = match sbody with
    | [] -> raise (Failure ("a function cannot have an empty body"))
    | _ ->
      let rec list_last list = match list with
        | [] -> failwith "List is empty"
        | [x] -> x
        | _::rest_of_list -> list_last rest_of_list
      in list_last sbody
  in

  let rtype = match check_empty with
    | SReturn(t, _) -> t
    | _ -> Void
  in
  (Func(param_types, rtype), rtype, sbody)
in


let rec oexpr vars = function
    OId(s) -> (type_of_identifier vars s, SOId s)
  | OBinop1(oe1, op, e2) ->
    let (ot1, oe1') = oexpr vars oe1
    and (t2, e2') = expr vars e2 in
```

```
      let t1 = get_observable_inner_type ot1 in
      let same = t1 = t2 in
      let ty = match op with
        | Add | Sub | Mult | Div when same && t1 = Int   -> Int
        | Add | Sub | Mult | Div when same && t1 = Float -> Float
        | Add | Sub | Mult | Div when same && t1 = Char -> Char
        | Equal | Neq            when same               -> Int
        | Less | Leq | Greater | Geq
                 when same && (t1 = Int || t1 = Float || t1 = Char) -> Int
        | And | Or when same && t1 = Int -> Int
        | Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater
        | Geq when ((t1 = Float && t2 = Int) || (t1 = Int && t2 = Float)) ->
Float
        | _ -> raise (Failure ("illegal binary operator "  ^
                               string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
                               string_of_typ t2))
      in (Observable(ty), SMap(
        (Func([t1],ty), SFuncExpr(
          [(t1, "x")],
          ty,
          [SReturn (
            ty,
            SBinop(
              (t1, SId("x")),
              op,
              (t2, e2')
            )
          )]
        )),
        (ot1, oe1')
      ))
    | OBinop2(e1, op, oe2) ->
      let (t1, e1') = expr vars e1
      and (ot2, oe2') = oexpr vars oe2 in

      let t2 = get_observable_inner_type ot2 in
      let same = t1 = t2 in
      let ty = match op with
        | Add | Sub | Mult | Div when same && t1 = Int   -> Int
```

```
        | Add | Sub | Mult | Div when same && t1 = Float -> Float
        | Add | Sub | Mult | Div when same && t1 = Char -> Char
        | Equal | Neq            when same               -> Int
        | Less | Leq | Greater | Geq
                  when same && (t1 = Int || t1 = Float || t1 = Char) -> Int
        | And | Or when same && t1 = Int -> Int
        | Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater
        | Geq when ((t1 = Float && t2 = Int) || (t1 = Int && t2 = Float)) ->
Float
        | _ -> raise (Failure ("illegal binary operator "  ^
                               string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
                               string_of_typ t2))
      in (Observable(ty), SMap(
        (Func([t2],ty), SFuncExpr(
          [(t2, "x")],
          ty,
          [SReturn (
            ty,
            SBinop(
              (t1, e1'),
              op,
              (t2, SId("x"))
            )
          )]
        )),
        (ot2, oe2')
      ))
    | OBinop3(oe1, op, oe2) ->
      let (ot1, oe1') = oexpr vars oe1
      and (ot2, oe2') = oexpr vars oe2 in

      let t1 = get_observable_inner_type ot1 in
      let t2 = get_observable_inner_type ot2 in
      let same = t1 = t2 in
      let ty = match op with
          Add | Sub | Mult | Div when same && t1 = Int   -> Int
        | Add | Sub | Mult | Div when same && t1 = Float -> Float
        | Add | Sub | Mult | Div when same && t1 = Char -> Char
        | Equal | Neq            when same               -> Int
        | Less | Leq | Greater | Geq
```

```
                when same && (t1 = Int || t1 = Float || t1 = Char) -> Int
        | And | Or when same && t1 = Int -> Int
        | Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater
        | Geq when ((t1 = Float && t2 = Int) || (t1 = Int && t2 = Float)) ->
Float
        | _ -> raise (Failure ("illegal binary operator "  ^
                               string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
                               string_of_typ t2))
      in (Observable(ty), SCombine(
        (Func([t1;t2],ty), SFuncExpr(
          [(t1, "x"); (t2, "y")],
          ty,
          [SReturn (
            ty,
            SBinop(
              (t1, SId("x")),
              op,
              (t2, SId("y"))
            )
          )]
        )),
        (ot1, oe1'),
        (ot2, oe2')
      ))
    | OUnop(op, oe) ->
      let (ty, oe') = oexpr vars oe in (ty, SOUnop(op, (ty, oe')))
    | Map(e, oe) ->
      let (t, e') = expr vars e in
      let (ot, oe') = oexpr vars oe in

      let (args, rt) = match t with
        | Func(args, rt) when rt != Void -> (args, rt)
        | _ as x-> raise (Failure ("illegal expression of type " ^ string_of_typ
x ^
                                   " with map()"))
      in

      let it' = match ot with
        | Observable x -> x
        | _ -> raise (Failure ("second arguement of map must be an observable"))
```

```
      in let _ = match args with
        | [a] when a = it' -> ()
        | _ -> raise (Failure ("map function type does not match"))
      in
      (Observable rt, SMap((t, e'), (ot, oe')))
    | Combine(e, oe1, oe2) ->
      let (t, e') = expr vars e in
      let (ot1, oe1') = oexpr vars oe1 in
      let (ot2, oe2') = oexpr vars oe2 in

      let (args, rt) = match t with
        | Func(args, rt) when rt != Void -> (args, rt)
        | _ as x-> raise (Failure ("illegal expression of type " ^ string_of_typ
x ^
                                   " with map()"))
      in

      let it1' = match ot1 with
        | Observable x -> x
        | _ -> raise (Failure ("second arguement of map must be an observable"))
      in let it2' = match ot2 with
        | Observable x -> x
        | _ -> raise (Failure ("third arguement of map must be an observable"))
      in let _ = match args with
        | [a;b] when a = it1' && b = it2' -> ()
        | _ -> raise (Failure ("map function type does not match"))
      in
      (Observable rt, SCombine((t, e'), (ot1, oe1'), (ot2, oe2')))
  in

  let check_obs_stmt vars = function
      Obs(t, e) -> SObs(t, e)
    | OExpr oe -> SOExpr(oexpr vars oe)
    | ODecl(lt, var, e) ->
      let (rt, e') = expr vars e in
      StringHash.add vars var lt;
      SODecl(lt, var, (rt, e'))
    | OODecl(lt, var, oe) ->
      let (rt, oe') = oexpr vars oe in
      StringHash.add vars var lt;
```

```
      SOODecl(lt, var, (rt, oe'))
    | OAssign(s, e) ->
      let (rt, e') = expr vars e in
      let lt = type_of_identifier vars s in
      SOAssign(lt, s, (rt, e'))
    | OOAssign(s, oe) ->
      let (ot, oe') = oexpr vars oe in
      let lt = type_of_identifier vars s in
      SOOAssign(lt, s, (ot, oe'))
    | Subscribe(e, oe) ->
      let (ft, e') = expr vars e in
      let (ot, oe') = oexpr vars oe in

      let (args, _) = match ft with
        | Func(args, rt) -> (args, rt)
        | _ as x-> raise (Failure ("illegal expression of type " ^ string_of_typ
x ^
                                   " with map()"))
      in

      let it' = match ot with
        | Observable x -> x
        | _ -> raise (Failure ("second arguement of map must be an observable"))
      in let _ = match args with
        | [a] when a = it' -> ()
        | _ -> raise (Failure ("map function type does not match"))
      in
      SSubscribe("subscribe", (ft, e'), (ot, oe'))
    | Complete(oe) ->
      let (ot, oe') = oexpr vars oe in
      let _ = match ot with
        | Observable x -> x
        | _ -> raise (Failure ("second arguement of map must be an observable"))
      in
      SComplete("complete", (ot, oe'))
    | _ -> raise (Failure ("Not Implemented 1000"))
  in


  let fdecl_to_assign_stmt vars func =
```

```
    let func_type = Func((List.map fst func.formals), func.typ) in
    StringHash.add vars func.fname func_type;

    let (func_type, rtype, sstmt) = check_func_decl vars (func.formals,
func.body) in

    SDecl(func_type, func.fname, (func_type, SFuncExpr(func.formals, rtype,
sstmt)))
  in

  let check_glob glob = match glob with
      Stmt stmt -> SStmt(check_stmt global_vars stmt)
    | Obs_Stmt obs_stmt -> SObs_Stmt(check_obs_stmt global_vars obs_stmt)
    | Fdecl func -> SStmt(fdecl_to_assign_stmt global_vars func)

  in (List.map check_glob globs)
```

8.1.1.11 ./testall.sh

```
#!/bin/sh

# Regression testing script for Seaflow
# Step through a list of files
#  Compile, run, and check the output of each expected-to-work test
#  Compile and check the error of each expected-to-fail test

# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the LLVM compiler
LLC="llc"

# Path to the C compiler
CC="cc"

# Path to the seaflow compiler.  Usually "./seaflow.native"
```

```
# Try "_build/seaflow.native" if ocamlbuild was unable to create a symbolic link.
SEAFLOW="./seaflow.native"

# Set time limit for all operations
ulimit -t 30

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

keep=0

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

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

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

# Run <args>
```

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

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

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

    echo -n "$basename..."

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

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.ll ${basename}.s ${basename}.exe
${basename}.out" &&
    Run "$SEAFLOW" "$1" ">" "${basename}.ll" &&
    Run "$LLC" "-relocation-model=pic" "${basename}.ll" ">" "${basename}.s" &&
    Run "$CC" "-o" "${basename}.exe" "${basename}.s" "utils.o" &&
    Run "./${basename}.exe" > "${basename}.out" &&
```

```
    Compare ${basename}.out ${reffile}.out ${basename}.diff

    # Report the status and clean up the generated files

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

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

    echo -n "$basename..."

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

    generatedfiles=""

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

    # Report the status and clean up the generated files

    if [ $error -eq 0 ] ; then
      if [ $keep -eq 0 ] ; then
          rm -f $generatedfiles
      fi
```

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

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

shift `expr $OPTIND - 1`

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

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

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

if [ $# -ge 1 ]
then
    files=$@
```

```
else
    files="test/*/test-*.flo test/*/fail-*.flo"
fi

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

exit $globalerror
```

8.1.1.12 ./utils.c

```
#include <stdio.h>

void array_concat(void **f, void **s, void **d)
{
  void **src_curr, **dest_curr;
  int i, j, len, flen, slen;
  char **src, **dest;

  flen = **(int **)f;
  slen = **(int **)s;
  len = slen + flen;

  for(i = 1; i <= flen; i++){
    src_curr = f + i;
```

```
    dest_curr = d + i;

    src = (char **) src_curr;
    dest = (char **) dest_curr;
    *dest = *src;
  }

  j = flen + 1;
  for (i = 1 ; i <= slen; i++) {
    src_curr = s + i;
    dest_curr = d + j++;

    src = (char **) src_curr;
    dest = (char **) dest_curr;
    *dest = *src;
  }
}

void print_string(void **s) {
  int len = **(int **)s;

  for(int i = 1; i <= len; i++){
    char **curr = (char **) s + i;
    printf("%c", **curr);
  }
  printf("\n");
}

int print_float(double d) {
  printf("%lf\n", d);
  return 0;
}

int print_char(char c) {
  printf("%c\n", c);
  return 0;
}

int print_int(int i) {
  printf("%d\n", i);
```

```
  return 0;
}
```

8.1.2 Test Code

8.1.2.1 ./test/array/fail-arr1.flo

```
int[] a = [1, 2, 3, 5];

float b = a[0];

printf(b);
```

8.1.2.2 ./test/array/fail-arr1.err

```
Fatal error: exception Failure("illegal assignment float = int in float b = a[0];
")
```

8.1.2.3 ./test/array/fail-arr2.flo

```
int[] a = [1, 2, 3, 5];

int[] a = [0, 2, 4, 6];

printi(a[0]);
```

8.1.2.4 ./test/array/fail-arr2.err

```
Fatal error: exception Failure("variable has already been assigned!")
```

8.1.2.5 ./test/array/fail-arr3.flo

```
int[] a = [1, 2, 'c', 5];

int b = a[0];

printi(b);
```

8.1.2.6 ./test/array/fail-arr3.err

```
Fatal error: exception Failure("array literal: type mismatch int != char")
```

8.1.2.7 ./test/array/test-arr-concat.flo

```
int[] a = [4,5,6];
int[] b = [7,8,9];

int[] c = a + b;

printi(c.length);

printi(c[0]);
printi(c[5]);

char[] d = ['a','b','c'];
char[] e = ['d','e','f'];
char[] f = d + e;
printi(f.length);
printc(f[0]);
printc(f[5]);

float[] fl = [1.3, 2.4, 5.6] + [0.3, 4.5];
printi(fl.length);
printf(fl[4]);
```

```
struct Test {
    int x;
    float y;
    char[] name;
};

struct Test one = {1,4.5, "hello"};
struct Test two = {3, 6.5, "world"};
struct Test three = {2,3.5, "good"};
struct Test four = {8, 16.5, "morning"};

struct Test[] t = [ one, two ];
struct Test[] t2 = [ three,four ];
struct Test[] t3 = t + t2;

printi(t3.length);
struct Test res = t3[2];
prints(res.name);
```

8.1.2.8 ./test/array/test-arr-concat.out

```
6
4
9
6
a
f
5
4.500000
4
good
```

8.1.2.9 ./test/array/test-arr-func.flo

```
(int)->(int) mult = (int y) -> {
        return y * 100;
};

(int)->(int) div = (int y) -> {
        return y / 100;
};

struct Foo {
    int x;
    (int)->(int) func;
};

struct Foo baz = { 15, div };

int quotient = (baz.func)(100);
printi(100 * quotient);
```

8.1.2.10 ./test/array/test-arr-func.out

```
100
```

8.1.2.11 ./test/array/test-arr-struct.flo

```
struct Point {
    int x;
    int y;
};

struct Point p = {1,2};
struct Point q = {3,4};

struct Point[] a = [p, q];

int func(struct Point[] b){
```

```
    struct Point d = a[1];
    return d.x;
}

printi(func(a));
```

8.1.2.12 ./test/array/test-arr-struct.out

```
3
```

8.1.2.13 ./test/array/test-arr1.flo

```
int[] a = [1, 2, 3, 5];

printi(a[0]);
printi(a[1]);
printi(a[2]);
printi(a[3]);
```

8.1.2.14 ./test/array/test-arr1.out

```
1
2
3
5
```

8.1.2.15 ./test/array/test-arr10.flo

```
int foo() {
    int[] a = [1, 2, 3, 5];
```

```
    return a[3];
}

printi(foo());
```

8.1.2.16 ./test/array/test-arr10.out

```
5
```

8.1.2.17 ./test/array/test-arr11.flo

```
float foo() {
    float[] a = [1.0, 3.4, 2.1];
    return a[1];
}

printf(foo());
```

8.1.2.18 ./test/array/test-arr11.out

```
3.400000
```

8.1.2.19 ./test/array/test-arr12.flo

```
char foo() {
    char[] a = ['a', 'b', 'c'];
    return a[1];
}

printc(foo());
```

8.1.2.20 ./test/array/test-arr12.out

```
b
```

8.1.2.21 ./test/array/test-arr13.flo

```
int foo() {
    int[] a = [1, 2, 3, 5];
    return a.length;
}

printi(foo());
```

8.1.2.22 ./test/array/test-arr13.out

```
4
```

8.1.2.23 ./test/array/test-arr14.flo

```
int foo(int[] a, int i) {
    return a[i];
}

int[] b = [5, 6, 7];

printi(foo(b, 0));
printi(foo(b, 1));
printi(foo(b, 2));
```

8.1.2.24 ./test/array/test-arr14.out

```
5
6
7
```

8.1.2.25 ./test/array/test-arr15.flo

```
float foo(float[] a, int i) {
    return a[i];
}

float[] b = [1.0, 3.4, 2.1];

printf(foo(b, 0));
printf(foo(b, 1));
```

8.1.2.26 ./test/array/test-arr15.out

```
1.000000
3.400000
```

8.1.2.27 ./test/array/test-arr16.flo

```
char foo(char[] a, int i) {
    return a[i];
}

char[] b = ['a', 'b', 'c'];
```

```
printc(foo(b, 0));
printc(foo(b, 1));
printc(foo(b, 2));
```

8.1.2.28 ./test/array/test-arr16.out

```
a
b
c
```

8.1.2.29 ./test/array/test-arr17.flo

```
int i = [1,2,3].length;
printi(i);
float f = [1.2, 4.3, 3.2][0];
printf(f);
int foo(int[] a) {
    return a[0];
}
int x = foo([15,2,3,4]);
printi(x);
```

8.1.2.30 ./test/array/test-arr17.out

```
3
1.200000
15
```

8.1.2.31 ./test/array/test-arr18.flo

```
(int)->(int) increment = (int y) -> {
        return y * 100;
};
(int)->(int) increment2 = (int y) -> {
        return y * 1000;
};
(int)->(int)[] arr = [increment, increment2];
int x = (arr[0])(1);
printi(x);
```

8.1.2.32 ./test/array/test-arr18.out

```
100
```

8.1.2.33 ./test/array/test-arr19.flo

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

int bar() {
    return 2;
}

int[] a = [1, 2, 3, 5];

printi(a[foo()]);
printi(a[bar()]);
```

8.1.2.34 ./test/array/test-arr19.out

```
1
3
```

8.1.2.35 ./test/array/test-arr2.flo

```
int[] a = [1, 2, 3, 5];
int y = 0;
int z = 2;

printi(a[y]);
printi(a[z]);
```

8.1.2.36 ./test/array/test-arr2.out

```
1
3
```

8.1.2.37 ./test/array/test-arr3.flo

```
float[] a = [1.0, 3.4, 2.1];

printf(a[0]);
printf(a[1]);
```

8.1.2.38 ./test/array/test-arr3.out

```
1.000000
3.400000
```

8.1.2.39 ./test/array/test-arr4.flo

```
float[] a = [1.0, 3.4, 2.1];
int y = 0;
int z = 1;

printf(a[y]);
printf(a[z]);
```

8.1.2.40 ./test/array/test-arr4.out

```
1.000000
3.400000
```

8.1.2.41 ./test/array/test-arr5.flo

```
char[] a = ['a', 'b', 'c'];

printc(a[0]);
printc(a[1]);
printc(a[2]);
```

8.1.2.42 ./test/array/test-arr5.out

```
a
b
c
```

8.1.2.43 ./test/array/test-arr6.flo

```
char[] a = ['a', 'b', 'c'];
```

```
int y = 0;
int z = 2;

printc(a[y]);
printc(a[z]);
```

8.1.2.44 ./test/array/test-arr6.out

```
a
c
```

8.1.2.45 ./test/array/test-arr7.flo

```
int[] a = [1, 2, 3, 5];
int[] copy = a;

printi(copy[0]);
printi(copy[1]);
printi(copy[2]);
printi(copy[3]);
```

8.1.2.46 ./test/array/test-arr7.out

```
1
2
3
5
```

8.1.2.47 ./test/array/test-arr8.flo

```
float[] a = [1.0, 3.4, 2.1];
float[] copy = a;

printf(copy[0]);
printf(copy[1]);
```

8.1.2.48 ./test/array/test-arr8.out

```
1.000000
3.400000
```

8.1.2.49 ./test/array/test-arr9.flo

```
char[] a = ['a', 'b', 'c'];
char[] copy = a;

printc(copy[0]);
printc(copy[1]);
printc(copy[2]);
```

8.1.2.50 ./test/array/test-arr9.out

```
a
b
c
```

8.1.2.51 ./test/array/test-arrstring.flo

```
char[] c = "hello world!";
printc(c[0]);
```

```
printc(c[1]);

printc(c[c.length - 1]);
```

8.1.2.52 ./test/array/test-arrstring.out

```
h
e
!
```

8.1.2.53 ./test/char/fail-char1.flo

```
char x = 'c';
char x = 'd';

printc(x);
```

8.1.2.54 ./test/char/fail-char1.err

```
Fatal error: exception Failure("variable has already been assigned!")
```

8.1.2.55 ./test/char/fail-char2.flo

```
int i = 7;
char j = 'd';

int k = i && j;

printi(k);
```

8.1.2.56 ./test/char/fail-char2.err

```
Fatal error: exception Failure("&& and || must be used with integers")
```

8.1.2.57 ./test/char/fail-char3.flo

```
char i = 'a';
char j = 'b';

int k = i && j;

printi(k);
```

8.1.2.58 ./test/char/fail-char3.err

```
Fatal error: exception Failure("&& and || must be used with integers")
```

8.1.2.59 ./test/char/fail-char4.flo

```
int i = 7;
char j = 'e';

int k = i || j;

printi(k);
```

8.1.2.60 ./test/char/fail-char4.err

```
Fatal error: exception Failure("&& and || must be used with integers")
```

8.1.2.61 ./test/char/fail-char5.flo

```
char i = 'a';
char j = 'b';

int k = i || j;

printi(k);
```

8.1.2.62 ./test/char/fail-char5.err

```
Fatal error: exception Failure("&& and || must be used with integers")
```

8.1.2.63 ./test/char/test-char1.flo

```
char i = '!';
char j = '#';
char k = i + j;

printc(i);
printc(j);
printc(k);
```

8.1.2.64 ./test/char/test-char1.out

```
!
#
D
```

8.1.2.65 ./test/char/test-char2.flo

```
void testChar(char a, char b) {
    int c = if(a == b) 1 else 0;
    printi(c);
    int d = if(a == a) 1 else 0;
    printi(d);
    int e = if(a != b) 1 else 0;
    printi(e);
    int f = if(a != a) 1 else 0;
    printi(f);
    int g = if(a > b) 1 else 0;
    printi(g);
    int h = if(a >= b) 1 else 0;
    printi(h);
    int i = if(a < b) 1 else 0;
    printi(i);
    int j = if(a <= b) 1 else 0;
    printi(j);
}

void foo() {
  char c = '!';
  char d = '#';

  testChar(c, d);
  testChar(d, d);
}

foo();
```

8.1.2.66 ./test/char/test-char2.out

```
0
```

```
1
1
0
0
0
1
1
1
1
0
0
0
1
0
1
```

8.1.2.67 ./test/float/fail-float1.flo

```
float x = 7.0;
float x = 12.0;

printf(x);
```

8.1.2.68 ./test/float/fail-float1.err

```
Fatal error: exception Failure("variable has already been assigned!")
```

8.1.2.69 ./test/float/fail-float2.flo

```
int i = 7;
float j = 12.0;

int k = i && j;
```

```
printi(k);
```

8.1.2.70 ./test/float/fail-float2.err

```
Fatal error: exception Failure("&& and || must be used with integers")
```

8.1.2.71 ./test/float/fail-float3.flo

```
float i = 7.0;
float j = 12.0;

int k = i && j;

printi(k);
```

8.1.2.72 ./test/float/fail-float3.err

```
Fatal error: exception Failure("&& and || must be used with integers")
```

8.1.2.73 ./test/float/fail-float4.flo

```
int i = 7;
float j = 12.0;

int k = i || j;

printi(k);
```

8.1.2.74 ./test/float/fail-float4.err

```
Fatal error: exception Failure("&& and || must be used with integers")
```

8.1.2.75 ./test/float/fail-float5.flo

```
float i = 7.0;
float j = 12.0;

int k = i || j;

printi(k);
```

8.1.2.76 ./test/float/fail-float5.err

```
Fatal error: exception Failure("&& and || must be used with integers")
```

8.1.2.77 ./test/float/test-float1.flo

```
float i = 7.0;
float j = 12.0;
float k = i + j;

printf(i);
printf(j);
printf(k);
```

8.1.2.78 ./test/float/test-float1.out

```
7.000000
12.000000
19.000000
```

8.1.2.79 ./test/float/test-float2.flo

```
void testFloat(float a, float b) {
    printf(a + b);
    printf(a - b);
    printf(a * b);
    printf(a / b);
    int c = if(a == b) 1 else 0;
    printi(c);
    int d = if(a == a) 1 else 0;
    printi(d);
    int e = if(a != b) 1 else 0;
    printi(e);
    int f = if(a != a) 1 else 0;
    printi(f);
    int g = if(a > b) 1 else 0;
    printi(g);
    int h = if(a >= b) 1 else 0;
    printi(h);
    int i = if(a < b) 1 else 0;
    printi(i);
    int j = if(a <= b) 1 else 0;
    printi(j);
}

void foo() {
  float c = 42.0;
  float d = 3.14159;

  testFloat(c, d);
  testFloat(d, d);
}

foo();
```

8.1.2.80 ./test/float/test-float2.out

```
45.141590
38.858410
131.946780
13.369027
0
1
1
0
1
1
0
0
6.283180
0.000000
9.869588
1.000000
1
1
0
0
0
1
0
1
```

8.1.2.81 ./test/float/test-float3.flo

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

```
    int c = if(a == b) 1 else 0;
    printi(c);
    int d = if(a == a) 1 else 0;
    printi(d);
    int e = if(a != b) 1 else 0;
    printi(e);
    int f = if(a != a) 1 else 0;
    printi(f);
    int g = if(a > b) 1 else 0;
    printi(g);
    int h = if(a >= b) 1 else 0;
    printi(h);
    int i = if(a < b) 1 else 0;
    printi(i);
    int j = if(a <= b) 1 else 0;
    printi(j);
}

void foo() {
  float c = 42.0;
  int d = 3;
  int e = 42;

  testFloatLeft(c, d);
  testFloatLeft(c, e);
}

foo();
```

8.1.2.82 ./test/float/test-float3.out

```
45.000000
39.000000
126.000000
14.000000
0
1
1
```

```
0
1
1
0
0
84.000000
0.000000
1764.000000
1.000000
1
1
0
0
0
1
0
1
```

8.1.2.83 ./test/float/test-float4.flo

```
void testFloatRight(int a, float b) {
    printf(a + b);
    printf(a - b);
    printf(a * b);
    printf(a / b);
    int c = if(a == b) 1 else 0;
    printi(c);
    int d = if(a == a) 1 else 0;
    printi(d);
    int e = if(a != b) 1 else 0;
    printi(e);
    int f = if(a != a) 1 else 0;
    printi(f);
    int g = if(a > b) 1 else 0;
    printi(g);
    int h = if(a >= b) 1 else 0;
    printi(h);
    int i = if(a < b) 1 else 0;
```

```
    printi(i);
    int j = if(a <= b) 1 else 0;
    printi(j);
}

void foo() {
  float c = 42.0;
  int d = 3;
  int e = 42;

  testFloatRight(d, c);
  testFloatRight(e, c);
}

foo();
```

8.1.2.84 ./test/float/test-float4.out

```
45.000000
-39.000000
126.000000
0.071429
0
1
1
0
0
0
1
1
84.000000
0.000000
1764.000000
1.000000
1
1
0
0
```

```
0
1
0
1
```

8.1.2.85 ./test/func/fail-func1.flo

```
int foo() {
  return 5;
}

int bar() {
  return 3;
}

int baz() {
  return 1;
}

void bar() {
  return 4;
} /* Error: duplicate function bar */

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

8.1.2.86 ./test/func/fail-func1.err

```
Fatal error: exception Failure("Cannot declare global more than once!")
```

8.1.2.87 ./test/func/fail-func10.flo

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

void foo() {
  int a = add(39, 'c');
  printi(a);
}

foo();
```

8.1.2.88 ./test/func/fail-func10.err

```
Fatal error: exception Failure("illegal argument: found char expected int in
'c'")
```

8.1.2.89 ./test/func/fail-func11.flo

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

void foo() {
  int q = 12;
  float r = 1.9;
  int a = add(q, r);
  printi(a);
}

foo();
```

8.1.2.90 ./test/func/fail-func11.err

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

8.1.2.91 ./test/func/fail-func12.flo

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

void foo() {
  float a = add(12, 7);
  printf(a);
}

foo();
```

8.1.2.92 ./test/func/fail-func12.err

```
Fatal error: exception Failure("illegal assignment float = int in float a =
add(12, 7);
")
```

8.1.2.93 ./test/func/fail-func2.flo

```
int foo(int a, char b, int c) {
  return 0;
}

int bar(int a, char b, int a) {
  return 0;
} /* Error: duplicate formal a in bar */

int baz() {
  return 0;
```

```
}
```

8.1.2.94 ./test/func/fail-func2.err

```
Fatal error: exception Failure("Cannot have duplicate formal!")
```

8.1.2.95 ./test/func/fail-func3.flo

```
int foo(int a, char b, int c) {
  return 7;
}

int bar(int a, void b, int c) {
  return 0;
} /* Error: illegal void formal b */

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

8.1.2.96 ./test/func/fail-func3.err

```
Fatal error: exception Failure("cannot have void formal!")
```

8.1.2.97 ./test/func/fail-func4.flo

```
int foo() {
  return 9;
}
```

```
int bar() {
  int a = 4;
  void b = 3; /* Error: illegal void local b */
  char c = 'c';

  return 0;
}

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

8.1.2.98 ./test/func/fail-func4.err

```
Fatal error: exception Failure("cannot assign to void type!")
```

8.1.2.99 ./test/func/fail-func5.flo

```
int foo(int a, char b) {
  return 2;
}

int bar() {
  foo(42, 'c');
  foo(42); /* Wrong number of arguments */
  return 0;
}

bar();
```

8.1.2.100 ./test/func/fail-func5.err

```
Fatal error: exception Failure("expecting 2 arguments in foo(42)")
```

8.1.2.101 ./test/func/fail-func6.flo

```
int foo(int a, char b) {
  return 0;
}

int bar() {
  foo(42, 'f');
  foo(42, 'e', 'f'); /* Wrong number of arguments */
}

bar();
```

8.1.2.102 ./test/func/fail-func6.err

```
Fatal error: exception Failure("expecting 2 arguments in foo(42, 'e', 'f')")
```

8.1.2.103 ./test/func/fail-func7.flo

```
int foo(int a, char b) {
  return 0;
}

int bar() {
  return 0;
}

int baz() {
  foo(42, 'a');
  foo(42, bar());
  return 8;
}
```

```
baz();
```

8.1.2.104 ./test/func/fail-func7.err

```
Fatal error: exception Failure("illegal argument: found int expected char in
bar()")
```

8.1.2.105 ./test/func/fail-func8.flo

```
int foo(int a, char b) {
  return 0;
}

int bar() {
  foo(42, 'c');
  foo(42, 42); /* Fail: int, not char */
}

bar();
```

8.1.2.106 ./test/func/fail-func8.err

```
Fatal error: exception Failure("illegal argument: found int expected char in 42")
```

8.1.2.107 ./test/func/fail-func9.flo

```
void foo() {}

int bar(int a, char b, int c) {
```

```
  return a + c;
}

void baz()
{
  printi(bar(17, 'd', 25));
}

baz();
```

8.1.2.108 ./test/func/fail-func9.err

```
Fatal error: exception Failure("a function cannot have an empty body")
```

8.1.2.109 ./test/func/test-func1.flo

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

void foo() {
  int a = add(39, 3);
  printi(a);
}

foo();
```

8.1.2.110 ./test/func/test-func1.out

```
42
```

8.1.2.111 ./test/func/test-func2.flo

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

void foo() {
  printem(42, 17, 192, 8);
}

foo();
```

8.1.2.112 ./test/func/test-func2.out

```
42
17
192
8
```

8.1.2.113 ./test/func/test-func3.flo

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

void foo() {
  int d = add(52, 10);
  printi(d);
}
```

```
foo();
```

8.1.2.114 ./test/func/test-func3.out

```
62
```

8.1.2.115 ./test/func/test-func4.flo

```
int foo(int a) {
  printi(12);
  return a;
}
```

8.1.2.116 ./test/func/test-func4.out

8.1.2.117 ./test/func/test-func5.flo

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

void bar() {
  foo(40);
}

bar();
```

8.1.2.118 ./test/func/test-func5.out

```
43
```

8.1.2.119 ./test/func/test-func6.flo

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

void bar() {
  int i = 40;
  foo(i);
}

bar();
```

8.1.2.120 ./test/func/test-func6.out

```
43
```

8.1.2.121 ./test/func/test-func7.flo

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

int foo() {
  return 7;
}

int bar() {
  return 8;
```

```
}

void baz() {
  int a = add(foo(), bar());
  printi(a);
}

baz();
```

8.1.2.122 ./test/func/test-func7.out

```
15
```

8.1.2.123 ./test/func/test-func8.flo

```
char foo(int a, char b) {
  return b;
}

void bar() {
  char d = foo(52, 'c');
  printc(d);
}

bar();
```

8.1.2.124 ./test/func/test-func8.out

```
c
```

8.1.2.125 ./test/func/test-func9.flo

```
int fib(int x) {
  int y = if (x > 2) fib(x-1) + fib(x-2) else 1;
  return y;
}

printi(fib(6));
```

8.1.2.126 ./test/func/test-func9.out

```
8
```

8.1.2.127 ./test/global/fail-global1.flo

```
int c = 5;
char b = 'c';
void a = 0; /* global variables should not be void */


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

8.1.2.128 ./test/global/fail-global1.err

```
Fatal error: exception Failure("cannot assign to void type!")
```

8.1.2.129 ./test/global/fail-global2.flo

```
int b = 7;
char c = 'd';
int a = 12;
int b = 2; /* Duplicate global variable */

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

8.1.2.130 ./test/global/fail-global2.err

```
Fatal error: exception Failure("variable has already been assigned!")
```

8.1.2.131 ./test/global/test-global1.flo

```
int i = 7;
printi(i);
```

8.1.2.132 ./test/global/test-global1.out

```
7
```

8.1.2.133 ./test/global/test-global2.flo

```
float f = 5.5;
printf(f);
```

8.1.2.134 ./test/global/test-global2.out

```
5.500000
```

8.1.2.135 ./test/global/test-global3.flo

```
char c = 'c';
printc(c);
```

8.1.2.136 ./test/global/test-global3.out

```
c
```

8.1.2.137 ./test/hof/fail-hof1.flo

```
(int)->(int) increment = (int x) -> {
    return x + 2;
};

int increment(int x) {
    return x + 1;
}

printi(increment(100));
```

8.1.2.138 ./test/hof/fail-hof1.err

```
Fatal error: exception Failure("Cannot declare global more than once!")
```

8.1.2.139 ./test/hof/test-hof1.flo

```
int some(int x) {
    return x + 1;
}

(int)->(int) increment = some;

((int)->(int), int)->(int) apply = ((int)->(int) f, int x) -> {
    (int)->(int) local = (int y) -> {
        return y * 100;
    };
    return local(f(x));
};

printi(apply(increment, 4));
```

8.1.2.140 ./test/hof/test-hof1.out

```
500
```

8.1.2.141 ./test/hof/test-hof2.flo

```
/* tests inline lambda expression */
int x = ((int x) -> {return x * 2;})(5);
printi(x);
```

8.1.2.142 ./test/hof/test-hof2.out

```
10
```

8.1.2.143 ./test/hof/test-hof3.flo

```
(int)->(int) mult = (int y) -> {
        return y * 100;
};

(int)->(int) div = (int y) -> {
        return y / 100;
};

struct Foo {
    int x;
    (int)->(int) func;
};

struct Foo baz = { 15, div };

int quotient = (baz.func)(100);
printi(quotient);
```

8.1.2.144 ./test/hof/test-hof3.out

```
1
```

8.1.2.145 ./test/if/fail-if1.flo

```
int foo() {
    printi(1);
    return 2;
}

int bar() {
    printi(0);
    return 3;
```

```
}

int y = if (1 == 1) foo() else bar();
printi(y);
```

8.1.2.146 ./test/if/fail-if1.err

```
Fatal error: exception Failure("Not Implemented 2002")
```

8.1.2.147 ./test/if/fail-if2.flo

```
int foo() {
    printi(1);
    return 2;
}

int bar() {
    printi(0);
    return 3;
}

float y = if (1 == 1) foo() else bar();
printi(y);
```

8.1.2.148 ./test/if/fail-if2.err

```
Fatal error: exception Failure("illegal if; types must match for lval and rval")
```

8.1.2.149 ./test/if/test-if1.flo

```
int foo() {
  int a = if (1 == 1) 42 else 17;
  printi(a);
  return 0;
}

foo();
```

8.1.2.150 ./test/if/test-if1.out

```
42
```

8.1.2.151 ./test/if/test-if2.flo

```
int foo() {
  int a = if (1 == 0) 42 else 17;
  printi(a);
  return 0;
}

foo();
```

8.1.2.152 ./test/if/test-if2.out

```
17
```

8.1.2.153 ./test/if/test-if3.flo

```
int foo(int i) {
    return i + 3;
```

```
}

int bar(int j) {
    return j + j;
}

int baz(int i, int j) {
  int a = if (1 == 1) foo(i) else bar(j);
  printi(a);
  return 0;
}

int q = baz(7, 12);
```

8.1.2.154 ./test/if/test-if3.out

```
10
```

8.1.2.155 ./test/if/test-if4.flo

```
int foo(int i) {
    return i + 3;
}

int bar(int j) {
    return j + j;
}

int baz(int i, int j) {
  int a = if (1 == 0) foo(i) else bar(j);
  printi(a);
  return 0;
}

int q = baz(7, 12);
```

8.1.2.156 ./test/if/test-if4.out

```
24
```

8.1.2.157 ./test/if/test-if5.flo

```
int foo() {
    printi(1);
    return 2;
}

int bar() {
    printi(0);
    return 3;
}

void baz() {
    int y = if (1 == 1) foo() else bar();
    printi(y);
}

baz();
```

8.1.2.158 ./test/if/test-if5.out

```
1
2
```

8.1.2.159 ./test/if/test-if6.flo

```
int foo() {
    printi(1);
    return 2;
}

int bar() {
    printi(0);
    return 3;
}

void baz() {
    int y = if (1 == 0) foo() else bar();
    printi(y);
}

baz();
```

8.1.2.160 ./test/if/test-if6.out

```
0
3
```

8.1.2.161 ./test/int/fail-int1.flo

```
int x = 7;
int x = 12;

printi(x);
```

8.1.2.162 ./test/int/fail-int1.err

```
Fatal error: exception Failure("variable has already been assigned!")
```

8.1.2.163 ./test/int/fail-int2.flo

```
int a = 7;

float b = a;

printf(b);
```

8.1.2.164 ./test/int/fail-int2.err

```
Fatal error: exception Failure("illegal assignment float = int in float b = a;
")
```

8.1.2.165 ./test/int/test-int1.flo

```
int i = 7;
int j = 12;
int k = i + j;

printi(i);
printi(j);
printi(k);
```

8.1.2.166 ./test/int/test-int1.out

```
7
12
19
```

8.1.2.167 ./test/int/test-int2.flo

```
void testInt(int a, int b) {
    printi(a + b);
    printi(a - b);
    printi(a * b);
    printi(a / b);
    int c = if(a == b) 1 else 0;
    printi(c);
    int d = if(a == a) 1 else 0;
    printi(d);
    int e = if(a != b) 1 else 0;
    printi(e);
    int f = if(a != a) 1 else 0;
    printi(f);
    int g = if(a > b) 1 else 0;
    printi(g);
    int h = if(a >= b) 1 else 0;
    printi(h);
    int i = if(a < b) 1 else 0;
    printi(i);
    int j = if(a <= b) 1 else 0;
    printi(j);
    int k = if (a && b) 1 else 0;
    printi(k);
    int l = if (a || b) 1 else 0;
    printi(l);
}

void foo() {
  int c = 42;
  int d = 3;

  testInt(c, d);
  testInt(d, d);
}

foo();
```

8.1.2.168 ./test/int/test-int2.out

```
45
39
126
14
0
1
1
0
1
1
0
0
1
1
6
0
9
1
1
1
0
0
0
1
0
1
1
1
```

8.1.2.169 ./test/local/fail-local1.flo

```
int foo(int a, char b, int c) {
```

```
  return 0;
}

int bar(int a, char b, int c) {
  int d = 0;
  int d = 7;
  return 0;
}

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

8.1.2.170 ./test/local/fail-local1.err

```
Fatal error: exception Failure("variable has already been assigned!")
```

8.1.2.171 ./test/local/fail-local2.flo

```
int foo(int a, char b, int c) {
  return 0;
}

int bar(int a, char b, int c) {
  void d = 0;
  return 0;
}

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

8.1.2.172 ./test/local/fail-local2.err

```
Fatal error: exception Failure("cannot assign to void type!")
```

8.1.2.173 ./test/local/test-local1.flo

```
void foo() {
    int i = 7;
    printi(i);
}

foo();
```

8.1.2.174 ./test/local/test-local1.out

```
7
```

8.1.2.175 ./test/local/test-local2.flo

```
void foo() {
    float f = 5.5;
    printf(f);
}

foo();
```

8.1.2.176 ./test/local/test-local2.out

```
5.500000
```

8.1.2.177 ./test/local/test-local3.flo

```
void foo() {
    char c = 'c';
    printc(c);
}

foo();
```

8.1.2.178 ./test/local/test-local3.out

```
c
```

8.1.2.179 ./test/observable/fail-global-obs1.flo

```
float apple(float x) {
    printf(x);
    return x;
}


int $a = 5;

subscribe(apple, $a);
```

8.1.2.180 ./test/observable/fail-global-obs1.err

```
Fatal error: exception Failure("map function type does not match")
```

8.1.2.181 ./test/observable/fail-global-obs2.flo

```
void apple(float x) {
    printf(x);
    return;
}

float multiple(int x, float y) {
    return x * y;
}


int $a = 5;
float b = 5.0;

float $c = combine(multiple, $a, b);
subscribe(apple, $c);

$a = 50;
```

8.1.2.182 ./test/observable/fail-global-obs2.err

```
Fatal error: exception Stdlib.Parsing.Parse_error
```

8.1.2.183 ./test/observable/fail-global-obs3.flo

```
int apple(int x) {
    printi(x);
    return x;
}

int increment(int x) {
    return x + 1;
}
```

```
int a = 5;
int $b = map(increment, a);
subscribe(apple, $b);
```

8.1.2.184 ./test/observable/fail-global-obs3.err

```
Fatal error: exception Stdlib.Parsing.Parse_error
```

8.1.2.185 ./test/observable/fail-global-obs4.flo

```
void apple(float x) {
    printf(x);
    return;
}

int a = 5;

complete(a);
```

8.1.2.186 ./test/observable/fail-global-obs4.err

```
Fatal error: exception Stdlib.Parsing.Parse_error
```

8.1.2.187 ./test/observable/fail-global-obs5.flo

```
float apple(float x) {
    printf(x);
    return x;
}
```

```
int a = 5;

subscribe(apple, a);
```

8.1.2.188 ./test/observable/fail-global-obs5.err

```
Fatal error: exception Stdlib.Parsing.Parse_error
```

8.1.2.189 ./test/observable/fail-global-obs6.flo

```
int apple(int x, int y) {
    printi(x);
    return x;
}


int $a = 5;
int $b = 6;

subscribe(apple, $a, $b);

$a = 4;
$a = 7;
```

8.1.2.190 ./test/observable/fail-global-obs6.err

```
Fatal error: exception Stdlib.Parsing.Parse_error
```

8.1.2.191 ./test/observable/fail-global-obs7.flo

```
void apple(float x) {
    printf(x);
    return;
}

float multiple(int x, float y) {
    return x * y;
}


int $a = 5;
float $b = 5.0;

float $c = combine(multiple, $a);
subscribe(apple, $c);

$a = 50;
```

8.1.2.192 ./test/observable/fail-global-obs7.err

```
Fatal error: exception Stdlib.Parsing.Parse_error
```

8.1.2.193 ./test/observable/test-global-obs1.flo

```
int apple(int x) {
    printi(x);
    return x;
}


int $a = 5;
```

```
subscribe(apple, $a);
```

8.1.2.194 ./test/observable/test-global-obs1.out

```
5
```

8.1.2.195 ./test/observable/test-global-obs10.flo

```
void apple(float x) {
    printf(x);
    return;
}

int $a = 5;
float $b = 5.0;

float $c = $a * $b;
subscribe(apple, $c);

$a = 50;
```

8.1.2.196 ./test/observable/test-global-obs10.out

```
25.000000
250.000000
```

8.1.2.197 ./test/observable/test-global-obs11.flo

```
void apple(float x) {
    printf(x);
```

```
    return;
}

int $a = 5;
float $b = 5.0;

float $c = $a * $b;
subscribe(apple, $c);

$a = 50;
complete($c);
```

8.1.2.198 ./test/observable/test-global-obs11.out

```
25.000000
250.000000
```

8.1.2.199 ./test/observable/test-global-obs12.flo

```
void apple(float x) {
    printf(x);
    return;
}

int $a = 5;
float $b = 5.0;

float $c = $a * $b;
subscribe(apple, $c);

$a = 50;
complete($a);
$a = 60;
```

8.1.2.200 ./test/observable/test-global-obs12.out

```
25.000000
250.000000
```

8.1.2.201 ./test/observable/test-global-obs13.flo

```
int apple(int x) {
    printi(x);
    return x;
}

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


int $a = 5;
int $b = $a - 1;

subscribe(apple, $b);

$a = 100;
```

8.1.2.202 ./test/observable/test-global-obs13.out

```
4
99
```

8.1.2.203 ./test/observable/test-global-obs14.flo

```
int apple(int x) {
    printi(x);
    return x;
}

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


int $a = 10;
int $b = 5;

int $c = $a / $b;

subscribe(apple, $c);

$a = 50;
$a = 500;
$b = 100;
```

8.1.2.204 ./test/observable/test-global-obs14.out

```
2
10
100
5
```

8.1.2.205 ./test/observable/test-global-obs2.flo

```
int apple(int x) {
    printi(x);
    return x;
}
```

```
int $a = 5;

subscribe(apple, $a);

$a = 4;
$a = 7;
```

8.1.2.206 ./test/observable/test-global-obs2.out

```
5
4
7
```

8.1.2.207 ./test/observable/test-global-obs3.flo

```
int apple(int x) {
    printi(x);
    return x;
}

int increment(int x) {
    return x + 1;
}


int $a = 5;
int $b = map(increment, $a);
subscribe(apple, $b);

$a = 4;
$a = 7;
```

8.1.2.208 ./test/observable/test-global-obs3.out

```
6
5
8
```

8.1.2.209 ./test/observable/test-global-obs4.flo

```
int apple(int x) {
    printi(x);
    return x;
}

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


int $a = 5;
int $b = 100;
int $c = combine(add, $a, $b);
subscribe(apple, $c);

$a = 100;
```

8.1.2.210 ./test/observable/test-global-obs4.out

```
105
200
```

8.1.2.211 ./test/observable/test-global-obs5.flo

```
int apple(int x) {
    printi(x);
    return x;
}

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


int $a = 5;
int $b = $a + 1;

subscribe(apple, $b);

$a = 100;
```

8.1.2.212 ./test/observable/test-global-obs5.out

```
6
101
```

8.1.2.213 ./test/observable/test-global-obs6.flo

```
int apple(int x) {
    printi(x);
    return x;
}

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

```
int $a = 5;
int $b = 10;

int $c = $a * $b;

subscribe(apple, $c);

$a = 50;
$a = 500;
$b = 100;
```

8.1.2.214 ./test/observable/test-global-obs6.out

```
50
500
5000
50000
```

8.1.2.215 ./test/observable/test-global-obs7.flo

```
int apple(int x) {
    printi(x);
    return x;
}

int $a = 5;
int $b = 10;

int $c = $a * $b;

subscribe(apple, $a);     /* 5         */
subscribe(apple, $c);     /* 50        */
```

```
$a = 50;                    /* 50, 500   */
$a = 500;                   /* 500, 5000 */
$b = 100;                   /* 50000     */
```

8.1.2.216 ./test/observable/test-global-obs7.out

```
5
50
50
500
500
5000
50000
```

8.1.2.217 ./test/observable/test-global-obs8.flo

```
int apple(int x) {
    printi(x);
    return x;
}

int $a = 5;

subscribe(apple, $a);     /* 5           */
subscribe(apple, $a);     /* 5           */
subscribe(apple, $a);     /* 5           */
subscribe(apple, $a);     /* 5           */
```

```
$a = 50;                    /* 50 50 50 50 */
```

8.1.2.218 ./test/observable/test-global-obs8.out

```
5
5
5
5
50
50
50
50
```

8.1.2.219 ./test/observable/test-global-obs9.flo

```
void apple(float x) {
    printf(x);
    return;
}

float multiple(int x, float y) {
    return x * y;
}


int $a = 5;
float $b = 5.0;

float $c = combine(multiple, $a, $b);
subscribe(apple, $c);

$a = 50;
```

8.1.2.220 ./test/observable/test-global-obs9.out

```
25.000000
```

```
250.000000
```

8.1.2.221 ./test/struct/fail-struct1.flo

```
void bar(int i, int j, char c) {
    struct Foo {
        int i;
        int j;
        char c;
    }; /* should fail, can only define structs on global scope */

    Foo foo = {i, j, c};
    printi(foo.i);
    printi(foo.j);
    printc(foo.c);
}

bar(4, 2, 'x');
```

8.1.2.222 ./test/struct/fail-struct1.err

```
Fatal error: exception Stdlib.Parsing.Parse_error
```

8.1.2.223 ./test/struct/test-struct1.flo

```
struct Point {
    int x;
    int y;
};

struct Point p = {4, 9};

printi(p.x);
```

```
printi(p.y);
```

8.1.2.224 ./test/struct/test-struct1.out

```
4
9
```

8.1.2.225 ./test/struct/test-struct2.flo

```
struct Complex {
    int i;
    char c;
    float f;
    int j;
    float g;
    char d;
};

struct Complex c = {9, 's', 1.0, 12, 2.0, 'c'};

printi(c.i);
printc(c.c);
printf(c.f);
printi(c.j);
printf(c.g);
printc(c.d);
```

8.1.2.226 ./test/struct/test-struct2.out

```
9
s
1.000000
12
```

```
2.000000
c
```

8.1.2.227 ./test/struct/test-struct3.flo

```
struct Foo {
    int i;
    int j;
    char c;
};

struct Foo foo = {4, 2, 'x'};

void bar(struct Foo f) {
    printi(f.i);
    printi(f.j);
    printc(f.c);
    return f.i;
}

bar(foo);
```

8.1.2.228 ./test/struct/test-struct3.out

```
4
2
x
```

8.1.2.229 ./test/struct/test-struct4.flo

```
struct Point {
    int x;
    int y;
```

```
};

struct Point2 {
    int z;
    int q;
};

struct Point p = {1,2};
struct Point2 q = p;
printi(q.z);
printi(q.q);
```

8.1.2.230 ./test/struct/test-struct4.out

```
1
2
```

8.1.2.231 ./test/struct/test-struct5.flo

```
struct Point {
    int x;
    int y;
};

struct Point2 {
    int z;
    int q;
};

int foo(struct Point p) {
    return p.x;
}

struct Point2 z = {3,4};
int x = foo(z);
```

```
printi(x);
```

8.1.2.232 ./test/struct/test-struct5.out

```
3
```

8.1.2.233 ./test/struct/test-struct6.flo

```
struct Foo {
    int i;
    int j;
    char c;
};

void bar(struct Foo f) {
    printi(f.i);
    printi(f.j);
    printc(f.c);
    return f.i;
}

bar({4, 2, 'x'});
```

8.1.2.234 ./test/struct/test-struct6.out

```
4
2
x
```

8.1.2.235 ./test/struct/test-struct7.flo

```
struct Point {
    int x;
    int y;
};
struct Point p = {1,2};
struct Point q = {3,4};
struct Point[] a = [p, q];
int func(struct Point[] b){
    struct Point d = a[1];
    return d.x;
}
printi(func(a));
```

8.1.2.236 ./test/struct/test-struct7.out

```
3
```

8.2 Git Log

```
commit 90faf7b0e4ee017e38a279dc97d4b4f00282d546 (HEAD -> master,
origin/master, origin/HEAD)
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Mon Apr 26 12:36:04 2021 -0400

      array concatenation works for any type

commit 9bc1e8b8c4f386647a5b4bf393fc74663573b92a
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Sun Apr 25 23:40:47 2021 -0400

      Update testall.sh

commit e617640c588a52d9e8406493dd5841412702e87d
```

```
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Sun Apr 25 23:39:54 2021 -0400

      Fix array offset

commit 965226ed330e5d913c5e3f3ddd2cda4687ecf289
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Sun Apr 25 22:22:29 2021 -0400

      cleaning up

commit e09ab68684f837f5e0330ecbb236bb8dbb351297
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Sun Apr 25 23:05:32 2021 +0000

      global obs fail

commit 0a073afa895d22eaf68d95b93426044b5f1a7998
Author: Junyang Jin <gallium@Junyangs-MacBook-Air.local>
Date:   Sun Apr 25 18:46:05 2021 -0400

      more details in dew-point demo

commit f35aa8b55884123a6707be92c033848e0814a0ed
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Sun Apr 25 18:27:53 2021 -0400

      Update test-func9.flo

commit 083725420b45de9ae0e2cd91dc784265dd565358
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Sun Apr 25 22:18:41 2021 +0000

      new fail if

commit fda296b0f036ece57bc58c5876aea0072acd54d7
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Sun Apr 25 17:50:04 2021 -0400
```

```
      recursive func demo added

commit abe6fbf92e8b2885ad2a5c09afb73ad7969f1aad
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Sun Apr 25 17:36:49 2021 -0400

      first order prints

commit 8d2b5ba1b244cfe8ce00392254e28628500226f3
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Sun Apr 25 21:15:28 2021 +0000

      observable divide and subtraction tests

commit 0fb2b95935f1e8bd08c54e21887046598a2692ca
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Sun Apr 25 17:03:24 2021 -0400

      Delete microc_tests directory

commit f915685235159fc54ad93fc8d55ced7912209296
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Sun Apr 25 17:02:54 2021 -0400

      Delete microcparse.mly

commit bfaba96ebe61280291be1694377a38195b3c931b
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Sun Apr 25 16:31:07 2021 -0400

      array null pointer setup

commit f2391d3a98c2694e6399176798ba8c3a44605d40
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Sun Apr 25 15:39:22 2021 -0400

      removed dangling files
```

```
commit 261e395e023c691c80241084ac800a8dac27fc68
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Sun Apr 25 15:26:09 2021 -0400

      Makefile update and warning fix

commit d7d5d965751b6a94977a26903e0d9ebc7c371525
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Sun Apr 25 18:34:04 2021 +0000

      arr

commit 0ccc3536098554631daf5198863f42c523b1d526
Merge: 2b3a067 d713142
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Sun Apr 25 18:31:17 2021 +0000

      merging

commit 2b3a06714d1845714636297171308e4221658a1f
Merge: 567c970 61022e1
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Sun Apr 25 18:27:04 2021 +0000

      merging

commit d7131427f1c982d6b0dd77aa57e5db69b98d9562
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Sun Apr 25 14:24:54 2021 -0400

      recursive call

commit 567c97003838a3cad7ed0f9384a8e998ffd3d6eb
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Sun Apr 25 18:24:04 2021 +0000

      fixed struct tests
```

```
commit 61022e147f17b516456ae7f6daff81f406c6a97f
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Sun Apr 25 13:59:51 2021 -0400

      concatenation for char arrays

commit ae0e5c9517ef5180e891a95e8b298cb6118a08af
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Sun Apr 25 13:41:13 2021 -0400

      array concat tests

commit 9389f388e5cd4c06929b8c7deffd223fec24a869
Merge: c87710d f4ab929
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Sun Apr 25 13:17:36 2021 -0400

      Merge pull request #5 from sarahseidman/array-concat

      Array concatenation works for integers!

commit f4ab92926af30d7c768bbe7b489c31ee12f8bf85
Merge: a69e236 c87710d
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Sun Apr 25 13:17:09 2021 -0400

      Merge branch 'master' into array-concat

commit a69e236497880f2a6c4233e294c538f6d95a8855
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Sun Apr 25 13:07:50 2021 -0400

      array concatenation :D

commit c87710d7d7c06255b6ae0f7310b36c21760c7ab1
Author: Junyang Jin <gallium@Junyangs-MacBook-Air.local>
Date:   Sun Apr 25 12:17:13 2021 -0400
```

```
      dew-point demo

commit 0ce5f227e60a0e273198b8034e9fb25bbca6823d
Merge: 3a9f4aa 0ef4aef
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Sun Apr 25 15:04:05 2021 +0000

      obs

commit 3a9f4aa3184024783a4b1e70dc0ecf515df2ecbf
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Sun Apr 25 14:59:59 2021 +0000

      local changes

commit 0ef4aef269b9f5a9fe0c6826977f1a36f3ae06da
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Sun Apr 25 10:51:09 2021 -0400

      resolved some compilation warnings

commit 5cc09978d223f358943e9edb616445d9520420b6
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Sat Apr 24 23:29:02 2021 -0400

      LBRAKT precedence fix

commit e546404ec1b19aa07507fefa57a7667ce8151727
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Sun Apr 25 03:22:06 2021 +0000

      hof again

commit f00b3ddabc87b8ffdc3e9f0bd6aeb05c2be0685f
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Sun Apr 25 03:18:26 2021 +0000
```

```
      hof

commit 878929e08dbe8a0a2787a789e3ae86ec01a69170
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Sun Apr 25 03:12:33 2021 +0000

      updated func and arr

commit 65866d48ae24267178e52ae25de2ad77e48b9a10
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Sun Apr 25 01:40:23 2021 +0000

      if tests moved and updated

commit d1a7ec79c527f1a3e208cd25957825c6ffd453b1
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Sat Apr 24 18:50:56 2021 -0400

      Create test-hof-4.out

commit 1d0bc870a242aa89390284ae84f4a3086b9c6f4f
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Sat Apr 24 18:50:41 2021 -0400

      Create test-hof-4.flo

commit 10c39c06854cf0a2a673fcec30747b5577c9dba6
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Sat Apr 24 22:40:45 2021 +0000

      array function test

commit 248ca9276155a512880321831ed8993d270d6fd7
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Sat Apr 24 22:18:47 2021 +0000

      new struct array test
```

```
commit 3b4caba5c83f4eee94b243d040bc711675682cad
Merge: cded026 bedb73f
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Sat Apr 24 22:15:46 2021 +0000

      merging

commit cded0265b1039ad8683e08c3544465bcb6b4d9bc
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Sat Apr 24 22:12:33 2021 +0000

      all tests

commit bedb73ff73b3cee0ac4aea55b85fe4347904f253
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Sat Apr 24 15:52:34 2021 -0400

      demos for struct and array

commit 5bc69e1dc7baec4629766e6f2776169a13a60c2a
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Sat Apr 24 15:52:18 2021 -0400

      null terminate string literals

commit 5e6d750be36b7d0259572e083123f22fd694f1a4
Merge: effee91 a495588
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Sat Apr 24 19:37:06 2021 +0000

      merging

commit effee91b167bc871e2b402bc40ad30e834ab6107
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Sat Apr 24 19:35:52 2021 +0000

      new tests again, changes to codegen for errors
```

```
commit a495588712df9d1d55f0f3a81ee1e729db11ece4
Merge: 3b913a8 e35b5d5
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Sat Apr 24 14:36:20 2021 -0400

      merge

commit 1a1db9c914cf41ff3d615b3f962fa61be7c9c3ca
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Sat Apr 24 14:29:41 2021 -0400

      working on array concat

commit e35b5d55aaa15a335a1a20f0c0192be3e3ba2273
Merge: 160859b f39dd1c
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Sat Apr 24 16:57:53 2021 +0000

      merging

commit 160859b074efe6f455815d0acb56c09f267bd7f4
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Sat Apr 24 16:56:00 2021 +0000

      new tests and error message

commit f39dd1c3d8679a4da823165981d6955edc233772
Author: Junyang Jin <gallium@Junyangs-MacBook-Air.local>
Date:   Sat Apr 24 12:25:29 2021 -0400

      complete method completed

commit 94ec0d2e000b5c3fb1a1f6843f88d0c07a8c3c25
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Sat Apr 24 11:16:25 2021 -0400

      && and || operators
```

```
commit 1a4fd839174a53aa2871acb972aa87ffab3013f5
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Sat Apr 24 11:01:54 2021 -0400

      array literal ops test

commit 3b913a8868ca0c3507d2598ff034c85d9fc4431c
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Sat Apr 24 10:40:35 2021 -0400

      array operations work on literals

commit 7fe3170c828e82246155aa8eba78fa86d63cc699
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Sat Apr 24 10:09:42 2021 -0400

      temp

commit f9fc5c949176876830168cb251a780ae7dea79a0
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Sat Apr 24 14:13:27 2021 +0000

      new tests

commit 5f84008976c387280d6a510476abe6e0304bd2fb
Author: Junyang Jin <gallium@Junyangs-MacBook-Air.local>
Date:   Sat Apr 24 09:24:23 2021 -0400

      preparation for complete method

commit 6e315a8f7ff500bb2da62de083d863f638bad800
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Fri Apr 23 23:57:05 2021 -0400

      SBinop fix

commit c8c4dd85fc87534a0a063a79c4d1a1233c2467e9
Author: Sanlok Lee <hl3436@columbia.edu>
```

```
Date:   Fri Apr 23 23:18:35 2021 -0400

      observable types

commit caef6cf62f7aff658e93df44da11b9f31d7ee78a
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Fri Apr 23 21:52:05 2021 -0400

      basic converter working

commit 1373310882ae517a24b59e89d2ebadc387d3a1e0
Merge: 01a05b8 bff0c50
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Sat Apr 24 00:31:54 2021 +0000

      merged in new if statement, new implicit conversion to double, new
error messages

commit bff0c50d23e4ac923dfdff7b58d9e35fc77acbd4
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Sat Apr 24 00:22:39 2021 +0000

      fixes

commit 114665f2392b083f94de6abaf021c8cf7db067d6
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Fri Apr 23 22:26:16 2021 +0000

      condition can be any type

commit f3348f1fc4b23dc0d8dd56c8c9891c396c32bd58
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Fri Apr 23 21:55:43 2021 +0000

      if statement working locally

commit 01a05b884f2fd8e7b0bd4bcb16ba72ca1845d619
Merge: 9ee1e73 374620d
```

```
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Fri Apr 23 17:07:07 2021 -0400

      Merge branch 'master' of https://github.com/sarahseidman/seaflow

commit 9ee1e73b7855db4264d7b45474971fb5595d8123
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Fri Apr 23 17:07:03 2021 -0400

      string literals to char array ; arrays now use void pointers

commit 374620d6620124f15639e7d011bcb8f855031db6
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Fri Apr 23 16:13:18 2021 -0400

      observable linkedlist multiple subscription

commit d07617de9ba2458ecdd4ed6ff11279c724a5169f
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Fri Apr 23 11:47:20 2021 -0400

      linkedlist with single subscription

commit 72e9d6eaba7df4c4adec3623d6e5a1638753e104
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Fri Apr 23 10:31:59 2021 -0400

      array length is no longer a special case of field reference

commit 2f8bd0f4c9954bf855ecab9e6aaf0ed1ef132830
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Fri Apr 23 04:54:40 2021 +0000

      working up to semant, need to get codegen working next

commit 4cd9683f26b53102031557cbb8336f78555597de
Merge: 1a94a21 0fa9621
Author: Rohan Arora <rarora9@yahoo.com>
```

```
Date:   Fri Apr 23 00:47:52 2021 +0000

      Empty func

commit 1a94a215e2b231dea369db87e49902019ee396f0
Merge: 1e2a055 2cf76f2
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Fri Apr 23 00:46:56 2021 +0000

      empty func

commit 0fa9621d1913cae4267f15bb7095799cfa7f4f53
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Thu Apr 22 20:45:25 2021 -0400

      Update codegen.ml

      2 small changes that got messed up in the merge

commit 2cf76f2f519aef30699b33e268f6c631f8ac57a3
Merge: adc4a6e a8718d2
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Thu Apr 22 20:18:28 2021 -0400

      Merge pull request #4 from sarahseidman/struct-refactor

      Struct refactor to use pointers

commit a8718d2ad6f2660f543ce4caa8fc64f13ddd8d16
Merge: 65d9a70 adc4a6e
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Thu Apr 22 20:18:12 2021 -0400

      Merge branch 'master' into struct-refactor

commit 65d9a70f325fe41941f2b24b259a78b568e1faf7
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Thu Apr 22 20:03:41 2021 -0400
```

```
      fix semantic checking

commit adc4a6eb875dc288a8b42f8b5a8aaa62faa4d54f
Merge: 087e3d6 d0c1c8a
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Thu Apr 22 20:01:57 2021 -0400

      Merge pull request #3 from sarahseidman/array

      Array

commit d0c1c8a1cc5051a9fa1e8c1c5f2ef0e8ccc5123e
Merge: 2fb065d 087e3d6
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Thu Apr 22 20:01:32 2021 -0400

      Merge branch 'master' into array

commit ea3283871871e65a13d47ba923db50118ed9ed37
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Thu Apr 22 18:46:55 2021 -0400

      structs work within functions

commit 087e3d6628660e2632ad02384a467267c7a55382
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Thu Apr 22 17:54:31 2021 -0400

      rebased master branch

commit b9ca68fc7c5d3485ea3666fc5df40e8a14d2e563
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Wed Apr 21 17:25:59 2021 -0400

      obinops implemented

commit 0e2355892ce469f7181eb7c90605d2625f51ef3b
```

```
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Wed Apr 21 16:16:13 2021 -0400

      combine implemented

commit 450433acaf856e058b669344a11b2767897a6492
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Wed Apr 21 00:32:49 2021 -0400

      map implemented with obs6

commit b93770e8c4d7d0984989a762a1a43ba72a40af18
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Tue Apr 20 23:53:25 2021 -0400

      recursive onnext fix

commit 738e2988fa5fd6325e9858140941f501a572209f
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Tue Apr 20 21:04:29 2021 -0400

      observable recursive

commit 1094ee74f24be2136c3b3ab6104f13af9ab60a7d
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Mon Apr 19 21:30:37 2021 -0400

      observable

commit c9bfa5be4588b6ed2e42f4077dd0cf92f98e7309
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Mon Apr 19 19:45:15 2021 -0400

      observables

commit 1e2a055bc2896f3d801fc116a34e05fef807a591
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Thu Apr 22 19:31:25 2021 +0000
```

```
      fixed empty funcs

commit 90ec88db36489f5200031913a40e6e4a3218c937
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Wed Apr 21 21:15:22 2021 -0400

      struct fix attempt

commit 3647fd1126eb281ee5a50851e1c30bae0e579e24
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Wed Apr 21 21:30:40 2021 +0000

      missing semicolon in a few array tests

commit b920b691d7b7d3c34c7c3aea913de15f072be3af
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Wed Apr 21 21:01:08 2021 +0000

      new tests and refactored folder structure

commit 24b6a0a54b4fdca80c3357f226fbcbe9454e47f5
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Wed Apr 21 20:47:27 2021 +0000

      implicit conversion from int to float for all ops

commit dc4f2f52bfaf1c9c04de2dfb47761213def6fb98
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Wed Apr 21 03:28:22 2021 +0000

      fixed float operations

commit 2fb065dd9afa24aa9da8b8e0771040817bc2b909
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Tue Apr 20 16:03:18 2021 -0400

      refactor arrays with pointers
```

```
commit 0880047b47e65b97efe58d03647ae0e550d73a7e
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Mon Apr 19 23:23:29 2021 +0000

      new test structure

commit f166f2e9e59c4e1d685dc587bdbc55bcf116d81c
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Sun Apr 18 10:47:27 2021 -0400

      malloc array test

commit 9c0a3009f74a608b5ca57b2c169f601573d2a965
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Fri Apr 16 20:50:11 2021 -0400

      array length

commit 4e8a7cff574b82a9bbff48c9fddced27ea5cdf6e
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Thu Apr 15 23:27:14 2021 -0400

      array decl/copying works within functions

commit e34102c012876882c9564f5c51bdcf1640c236ed
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Thu Apr 15 11:39:08 2021 -0400

      array copying

commit 45af61ee4ae07158b01114a43a1491300950d942
Author: Rohan Arora <rarora9@yahoo.com>
Date:   Thu Apr 15 15:32:48 2021 +0000

      fix out file

commit 14e2651017b23f7c67a5b11a20368cdbb94edcf8
```

```
Author: rarora9 <rarora9@yahoo.com>
Date:   Thu Apr 15 10:20:45 2021 -0400

      struct tests

commit abfcd1b77a2ba59ce68e05afd01463feae1648e7
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Wed Apr 14 20:39:45 2021 -0400

      array reference

commit f50d85cfbab05d4550f854e77308fade0183e05d
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Wed Apr 14 17:31:31 2021 -0400

      array declaration

commit 6175c62433eac075ba7c4a71faf18bac6ea9293a
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Wed Apr 14 17:30:38 2021 -0400

      rename microc test directory

commit bf59af412e8809bdb73513f10e9f26bd09365812
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Wed Apr 14 10:19:38 2021 -0400

      fixed gitignore

commit b25defec8c6aa837e733ac032eeda062ac6e2dba
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Wed Apr 14 09:45:08 2021 -0400

      makefile and ignore update

commit 2cee592590ffed6ad212a2ffb0f4715895417d8f
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Wed Apr 14 09:24:58 2021 -0400
```

```
      FuncExpr support in SCall

commit e3c7a9e44f417df3b94fd06f5ed5c9ed1eeedf54
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Tue Apr 13 21:15:19 2021 -0400

      hof implemented

commit 54603c9d7e32aba51a02837edb3c49d58f8a21a1
Merge: 853fb1c 0b31f46
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Tue Apr 13 22:21:58 2021 -0400

      Merge pull request #2 from sarahseidman/if

      Struct implementation

commit 0b31f4671e02a3887a4b1c30bd3539db53efd790
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Tue Apr 13 22:03:29 2021 -0400

      struct works in generalized case

commit 1dd63f95dbd4f58f59cac86952d3221523c8cfc0
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Tue Apr 13 21:36:53 2021 -0400

      struct is working

commit 853fb1cfc8c7b5dd97b70bc9c2094af6e255055a
Author: rarora9 <rarora9@yahoo.com>
Date:   Tue Apr 13 19:59:41 2021 -0400

      if tests

commit 70198f3c4302b86181e4ba2424d22025aafee0e5
Author: Sarah Seidman <ss5311@barnard.edu>
```

```
Date:   Mon Apr 12 17:50:13 2021 -0400

      struct but with segfault

commit 4efb621cae4387e1da37105b2248dbf44eedb0fb
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Mon Apr 12 15:22:56 2021 -0400

      struct sast

commit a02a2811796b122e39faae7b532bf204f5853110
Merge: 0db17b7 bff43fe
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Thu Apr 8 18:49:08 2021 -0400

      Merge pull request #1 from sarahseidman/if

      If expressions

commit bff43fe990bed4ddc89793d54fbae7974513fd78
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Mon Apr 5 21:26:22 2021 -0400

      if expressions

commit 972455fa09d53d2dda9c898e3539e23a989e2cfe
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Mon Apr 5 18:52:38 2021 -0400

      started working on if

commit 0db17b7af9cb6eefe33832bbdaecb41db6bf7298
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Sun Apr 4 14:40:10 2021 -0400

      add subtraction, multiplication to parser

commit 4092edd6507b5b20434714b2774977671676ec8d
```

```
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Sun Apr 4 14:14:25 2021 -0400

      function definition and call

commit 6e5864ee14d6a2e5865631cb338e10a67fc2e684
Author: rarora9 <rarora9@yahoo.com>
Date:   Sat Apr 3 12:49:57 2021 -0400

      func fail and pass tests and global var fails

commit 762d85d8eb3ee02af31a5a5d9d12dc2eac4fa44d
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Fri Apr 2 11:56:51 2021 -0400

      pretty-printing functions for SAST

commit 4d2c5a85a16e3ecbe3df3613a180654b8eadb7dd
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Mon Mar 29 22:31:34 2021 -0400

      basic function

commit efea949595e97a98f05229b0567138e169ce0eb8
Author: rarora9 <rarora9@yahoo.com>
Date:   Sun Mar 28 20:01:45 2021 -0400

      global vars and global observables tests

commit 6348bfd818d7bbfbf6bd3c3bac2e49df8b1d8006
Merge: df51c0d b9b2164
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Thu Mar 25 20:34:21 2021 -0400

      Merge branch 'master' of https://github.com/sarahseidman/seaflow

commit df51c0d7424e0a042976f840185965c2d71f4b92
Author: Sarah Seidman <ss5311@barnard.edu>
```

```
Date:   Fri Mar 26 00:33:27 2021 +0000

      switch to StringHash

commit b9b2164d34c291f56c7c3dba1debda21846a7e6d
Author: rarora9 <rarora9@yahoo.com>
Date:   Wed Mar 24 22:16:14 2021 -0400

      tests for var decls ops and local vars

commit 88c8022964d4efe6102d7f11325f00d3ba26954d
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Tue Mar 23 18:31:57 2021 +0000

      pretty-printing functions for AST

commit 54e6d12f9d53e85ab75dc69e046594068e910b53
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Mon Mar 22 21:23:02 2021 -0400

      hello world implemented

commit ca163a1eb8b969887d69e3da2f7fc22a5077cf5c
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Sat Mar 20 15:43:22 2021 -0400

      started implementing semant.ml

commit 30e3e2dedbe045baf1d8f1f9b40ac7c00f8907b0
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Wed Mar 17 21:27:22 2021 -0400

      ast.ml implemented

commit 1a0d05b63abeef50dcd11cd39a8d2c9d7000c562
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Mon Mar 15 22:26:09 2021 -0400
```

```
      compiler

commit c0a610233c1438b60b4c62ec17b77f85db789734
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Sat Mar 13 15:03:07 2021 -0500

      ast.ml implemented

commit 5b5570429bbd8ff8c77f7aa5ea9c6b249a7ce484
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Sat Mar 13 13:01:36 2021 -0500

      seaflowparse.mly implemented

commit f766fb3c15b36d1173ad3f9c1ae8dc322657a5ba
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Tue Feb 23 10:34:55 2021 -0500

      STRUCT SID fix 2

commit 035568da11f7b094e38a7a2895eb1b3411922bea
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Tue Feb 23 10:21:07 2021 -0500

      STRUCT SID fix

commit e981a3a4c0ecfc4811192e88c3ba6b4e1625b3ce
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Mon Feb 22 22:42:05 2021 -0500

      partial parser implementation

commit 3a5f5b75697e705e339d4697d546f2068383c6c3
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Sun Feb 21 21:18:05 2021 -0500

      parser implementation partially completed
```

```
commit 2aafd8295f85cef29596571422ddf235c5f23d46
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Sun Feb 21 19:14:52 2021 -0500

      Dockerfile fix 2

commit b1668c6832c39fba1db5e412abcbf541cab67da3
Author: Sanlok Lee <hl3436@columbia.edu>
Date:   Sat Feb 20 12:56:01 2021 -0500

      Dockerfile fix

commit 9aa4d01678fe32abd81c3784812a72382ccd8a5f
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Sat Feb 20 14:16:19 2021 -0500

      remove bool, add char and observable

commit 3142f601acc529c4e9a7e0f7487933bb9d44e720
Author: Sarah Seidman <ss5311@barnard.edu>
Date:   Thu Feb 18 22:21:47 2021 -0500

      microc
```