GaE Final Report

Graphs ain't Easy: A Graphing Language

Kevin Zeng (UNI: ksz2109)
Andrew Jones (UNI: adj2129)
Samara Nebel (UNI: srn2134)

Table of Contents

---

    f. Project Log
5. Architectural Design
    a. Diagram
    b. The Interfaces
    c. Implementers and Attribution
6. Test Plan
    a. Test Suites
    b. Test Automation
    c. Example Programs
7. Lessons Learned
    a. Andrew Jones
    b. Samara Nebel
    c. Kevin Zeng
8. Appendix
    a. Code Listing

Introduction

Graphs are prevalent in many different places across the internet. For example, every social media platform has graphs representing friends (e.g. Facebook) or followers (e.g. Twitter). The goal of this language is to allow the programmer to create a language that provides a much easier way to create and manipulate graphs with any data type stored inside the nodes, including custom structs, compared to conventional programming languages. With easy to use syntax, Dijkstra's algorithm, Kruskal's algorithm, and other commonly used graph algorithms can be easily implemented. Any program that relies on a graph data structure to hold and manipulate data and involves complex graph traversal and manipulation will be ideal for this language. The language is named GaE — Graphs Ain't Easy — in the spirit of the difficulty of writing programs that use graphs, but the language aims to make the experience much easier.

Language Tutorial

Hello World and Compiler Usage

Once you have downloaded the GaE tar.gz file, unzip the folder and enter the GaE directory. Type `make` to create the GaE compiler. This will take GaE code and convert it into a machine-readable executable. Once the GaE compiler has been made, programs can be compiled into an executable by running the `compile.sh` shell script. The following sample code is the implementation of Hello World in GaE.

```
1  func main() int {
2
3    prints("Hello World!");
4  }
5
```

Enter the above code into a file named helloworld.gae. To obtain the compiler executable from the language source code, type:

```
> make
```

To compile and run this code, type:

```
> ./compile.sh helloworld.gae
> ./helloworld.gae.out
```

We will now go through this code line by line.

• GaE files need to have a `func main() int` function. This is where program execution occurs. Functions in GaE must have return types. By default, main returns an `int` type; this is the `int` in `func main()int`. The curly brace that follows this header tells the program how the following code relates to the previous code.

• `prints` is a built-in function, meaning that it can be called in any GaE file without having to define it separately. This function prints a `string`. Other types have their own built-in print functions (e.g. `printi` for integers, `printb` for booleans, etc.) We pass the string “Hello World” into `prints`. The `string` type is indicated by the quotation marks. Note that this line

ends with a semicolon. All statements in GaE must end in a semicolon or in a curly brace. Semicolons are used to tell the compiler when a line ends.

• Finally, we close out with a closing curly brace to tell the program that main has ended.

We will now examine how we ran the code.

• Once we have our helloworld.gae file, we need to compile it. We use compile.sh to turn our GaE code into an executable and then run the executable.

And that's it! Congrats on writing your first piece of GaE code!

Language Reference Manual

Lexical Conventions

Tokens

There are 5 classes of tokens: identifiers, keywords, operators, literals, and separators. Blanks, horizontal and vertical tabs, newlines, and comments (described below) are ignored when used on their own, though white space is required to separate adjacent keywords and identifiers.

Comments

The characters /* introduce a comment which terminates with the characters */. These comments can span multiple lines, but do not nest, and do not occur within string or character literals.

Identifiers

An identifier consists of a sequence of characters that can be a letter, underscore, or digit. The first character of a variable identifier must be a lowercase letter (e.g. foo), while the first character of a custom struct identifier must be an uppercase letter (e.g. Integer). Identifiers are case sensitive.

Keywords

The following identifiers are reserved for use as keywords and cannot be used otherwise:

return	if	else	char
for	while	double	map
in	int	struct	edge
bool	string	true	
graph	func	false	

Literals

String Literals

A sequence of characters inside of double quotes.

e.g. `"string"`

Character Literals

A single character inside of single quotes.

e.g. `'a'`

Boolean Literals

A boolean literal is either `true` or `false`.

Integer Literals

A sequence of one or more digits representing an unnamed integer. Cannot start with 0.

e.g. `1337`

Double Literals

A double literal consists of an integer part, a decimal point, a fraction part, an `e` or an `E`, and an optionally signed integer component. Either the integer part or fraction part (not both) may be missing; either the decimal point or the e and the exponent (not both) may be missing.

e.g. `5.4e-15`

Struct Literals

A struct literal consists of a field and value pair specified by a field name, a colon, and a value. It can contain any number of field and value pairs, and the whole thing is encapsulated by curly brackets. All fields must be unique. You cannot have an empty struct literal.

e.g. `{value: 5}`

Array Literals

An array literal is a sequence of values (of the same type), separated by commas and encapsulated by square brackets.

e.g. `[1, 2, 3, 4, 5]`

Map Literals

A map literal is a key and value pair separated by a colon, and encapsulated by square brackets. All keys must be the same type, and all values must be the same type, though the key type does

not have to be the same as the value type. A key type must be integer, character, or a string. All keys must be unique.

e.g. `["foo":5, "bar": 8]`

Edge Literals

An edge literal is encapsulated by parentheses, and consists of a three-tuple that specifies the source node, destination node, and edge weight (in that order) with struct literals. The source and destination nodes must have the same struct type.

e.g. `({value: 1}, {value: 2}, {value: 3})`

Graph Literals

A graph literal is encapsulated by curly brackets and consists edge literals. All nodes must have the same struct type (i.e. they must have the exact same fields), and all edge weights must have the same struct type, though nodes and edge weights don't need to have the same struct type. Each edge literal must have a unique source-destination combination such that no two edge literals have the same source and destination. You cannot have an empty graph literal.

e.g.
```
{
        ({value: 1}, {value: 2}, {value: 3}),
        ({value: 2}, {value: 3}, {value: 1})
    }
```

Separators

The following characters are considered separators, i.e. characters that separate tokens

`( ) [ ] { } ; , :`

Semicolons are used exclusively for terminating expressions. Use of the other separators will be defined in the following sections.

Data Types

Primitive Data types

Integer

An integer is a number of the set $\mathbb{Z}$ = [-2147483648, 2147483647]. Integers are 32-bit signed integers that can be specified in decimal (base 10). The negation operator can be used to denote a negative integer. The type is denoted with `int`.

e.g.
```
int i;
i := 101;
```

Double

A double is a 32-bit floating-point number that can be specified using the following syntax. The type is denoted with `double`.

e.g.
```
double i;
double := 2.4;
```

Boolean

A boolean expresses a truth value. It can be either true or false. To specify a boolean literal, use the key words true or false. Both literals are case-sensitive.

e.g.
```
bool lies;
lies := true;
```

Char

A `char` is an ASCII character that can be any letter, number, punctuation marks, symbols or whitespace. A variable can be declared as a character type and characters can be stored in the variable. A char literal can only be specified using single quotes.

e.g.
```
char eye;
char := 'i';
```

String

A string is an array of characters terminated with a special character '\0'. A string literal can only be specified using double quotes.

e.g.
```
string greeting;
greeting := "Hello World";
```

Container Data Types

Structs

A struct is a user-defined data type that, unlike arrays, allows for the combination of data items of different kinds. A struct definition is as follows

e.g.
```
struct Int {
    value: int
}
```

This defines a custom struct type that we can use to define variables. For example, given the above struct definition, we can have:

e.g.
```
Int foo;
foo := {value : 5};
```

Array

An array is a container of one or more values of the same type. Each value is called an element of the array. The elements of the array share the same variable name. Each element has its own unique index number. An array can be of any primitive type (i.e. ints, doubles, strings, chars, and booleans) or structs.

e.g.
```
string[] weekdays;
weekdays := ["Saturday", "Sunday"];
double[] average;
average := [85.6, 91.7];
```

Map

A map is a container of key-value pairings, where a value can be accessed given a key but not vice-versa. The type of the key and value are defined by the type definition on the left-hand side of the declaration.

e.g. `map<string, double> gpas := ["andrew": 4.0, "kevin": 4.0, "samara": 4.0];`

Edge

An edge is a three-tuple consisting of source node, destination node, and edge weight. Each element of the tuple must be a struct. The node type and edge weight type must be declared.

e.g.
```
edge<Int, Int> some_edge;
Some_edge := ({value: 1}, {value: 2}, {value: 3});
```

Graph

A graph is a collection of edges. The node type and edge weight type must be defined on the left-hand side of the declaration.

e.g.
```
graph<Int, Int> some_graph;
   some_graph := {
                ({value: 1},{value: 2},{value: 3}),
                ({value: 2},{value: 1},{value:5})
                  };
```

Operators

Boolean Operators

||
This is the logical OR operator. The || operator guarantees left-to-right evaluation: the first operand is evaluated; if it is unequal to false, the value of the expression is true. Otherwise, the right operand is evaluated, and if it is unequal to false, the expression's value is true, otherwise false. The operands must both have type bool. The result is type `bool`.

e.g.

```
true || false /* returns true */
false || false /* returns false */
false || true /* returns true */
```

&&
This is the logical AND operator. The && operator guarantees left-to-right evaluation: the first operand is evaluated; if it is equal to false, the value of the expression is false. Otherwise, the right operand is evaluated, and if it is equal to false, the expression's value is false, otherwise true. The operands must both have type bool. The result is type `bool`.

e.g.

```
false && true /* returns false */
true && false /* returns false */
true && true /* returns true */
```

!
This is the logical NOT operator. The ! operand of the operator must have an boolean type. The operator negates the boolean value of the operand. The result is type `bool`.

e.g.

```
!true /* returns false */
!false /* returns true */
```

Numeric Operators

<, >, <=, >=
These are the RELATIONAL operators. These operators evaluate from left-to-right: if the relation is true, the operator returns true; and if the relation is false, the operator returns false.

The result is type `boolean`. These operators compare numeric values, not pointers. The operands must have the same type (and only supports ints and doubles).

e.g. `1 > 2 /* returns false */`

==, !=

These are the EQUALITY operators. These operators are analogous to the relational operators, except they have lower precedence. Thus, a<b == c<d would be evaluated: (a<b) == (c<d) where the operands on either side of the equality operators are evaluated first. The operands must have the same type (and only supports ints, doubles, strings, and structs).

e.g. `1 == 1 /* returns true */`

e.g. `2 == 5 /* returns false */`

Integer Operators

+, -, *, /, %, ++, --

These are the INTEGER operators. The **+** operator denotes addition, the **-** operator denotes subtraction, the ***** operator denotes multiplication, the **/** operator denotes division, the **%** operator yields the remainder, the **++** operator increments, and the **--** operator decrements. The result is type `int`. All of these operators may be used *only* in **integer** arithmetic; if used incorrectly, the compiler will throw a "type mismatch" error.

e.g.

```
5 + 6 /* returns 11 */
5.0 + 6 /* error */
```

String Operators

+

This operator concatenates two strings. Note the overloading of + operator (used for integers as well).

e.g. `"hello" + " world" == "hello world"; /* returns true */`

Double Operators

+., -., *., /., %.

These are the DOUBLE operators. The **+.** operator denotes addition, the **-.** operator denotes subtraction, the ***.** operator denotes multiplication, the **/.** operator denotes division, and the **%.** operator yields the remainder. The result is type `double`. All of these operators may be used *only* in **double** arithmetic; if used incorrectly, the compiler will throw a "type mismatch" error.

e.g.

```
5.0 +. 6.0 /* returns 11.0 */
5.0 +. 6 /* error */
```

Assignment Operators

:=, =

Formally, := is the ASSIGNMENT operator used the first time you assign a value to a variable. The = operator can then be used to reassign a variable. In practice, using either operator will exhibit the same outcomes. The := and = operators evaluate from right-to-left and the value of the expression replaces the object referred to by the left operand. Both operands must have the same type. The type of the left operand precedes the variable name, and the type of the right operand is inferred based on the syntax.

e.g.

```
int i;
i := 2;
i = 3;
```

Index Operator [ ]

Array

`array[i]`

Returns the ith element in the array (starts at index 0). If there is no ith element, throw an error.

e.g.
```
int[] arr;
   arr := [1,2,3,4];
   arr[1]; /* returns int of value 2 */
```

Map

`map[k]`

Returns the value mapped to given key k. If key k doesn't exist in map, throw an error.

e.g.
```
map<int, int> mymap;
   mymap := [1:2,3:4,5:6];
   mymap[3]; /* returns int of value 4 */
```

The `in` Operator

The `in` operator is evaluated from left to right and takes two arguments in the form

```
arg1 in arg2
```

and returns a boolean value for whether `arg1` is contained by `arg2`. Note that use of this operator is valid only when `arg2` is an array, map or graph. With graphs, `arg1` must be the same struct type as a graph node. Here is how the operator's behavior is defined in each case:

Array

When `arg2` is an array, `arg1 in arg2` returns true if and only if `arg1` is an element of `arg2`.

e.g. `5 in [1,2,3,4,5] /* returns true */`

Map

When `arg2` is a map, `arg1 in arg2` returns true if and only if `arg1` is a key of `arg2`.

```
e.g. 5 in [1:2,3:4,5:6] /* returns true */
     6 in [1:2,3:4,5:6] /* returns false */
```

Graph

When `arg2` is a graph, `arg1 in arg2` returns true if and only if `arg1` is a node of `arg2`.

```
e.g. {value: 5} in {({value: 1}, {value:5}, {value: 3})}
     /* returns true */
```

Control flow

If-statements

If-statements are used to define a block of code whose execution is conditional based on an expression that returns a boolean value. They follow the form

```
if expr {
     /* some code */
}
```

They may optionally contain an else-statement, which is executed in the event that expr evaluates to false:

```
if expr {
     /* some code */
} else {
     /* some more code */
}
```

Here is an example piece of code making use of if statements.

```
int foo;
foo := 5;
if foo > 10 {
     print(“your number is greater than 10”);
}
else {
     print(“your number is less than or equal to 10”);
}
```

Loops

There are two kinds of loops we support: while-loops and for-loops. While-loops define a block of code that is repeatedly executed as long as a given expression continues to evaluate as true. While-loops follow the form

```
while expr {
     /* some code */
}
```

An example while-loop that prints the numbers 1 to 10 is as follows:

```
int x;
x := 1;
while x <= 10 {
     print(x);
     x++;
}
```

For-loops are similar to while-loops in that they define a block of code that is repeatedly executed while a given expression continues to evaluate as true, but there is syntactic support for variable instantiation and modification after every execution of the defined block of code. For-loops follow the form:

```
for initialize; test; step {
     /* some code */
}
```

An example for-loop that prints the odd numbers from 1 to 20 is as follows:

```
int x;
for x := 1; x < 20; x=x+2 {
     print(x);
}
```

We also support nested loops, where a while or for-loop is embedded within an outer while or for-loop. An example of nesting loops to find all prime numbers under 100 is as follows:

```
int x;
int y;
bool is_prime;
for x := 2; x < 100; x++ {
     is_prime := true;
     for y := 2; y < x; y++ {
          if x % y == 0 {
               is_prime = false;
          }
     if is_prime {
          print(x);
     }
}
```

Functions

A function declaration has the form:

```
func func_name(parameter-list) return-type
```

The parameter list specifies the types of the parameters, which are separated by commas. A function is then followed by curly brackets. Inside the curly brackets is the code for the function, which must include a return statement denoted by the `return` keyword, returning the correct type as specified in the function declaration. All functions must specify a return type. All programs must have a main function (`func main() int`) which must follow this format. The code within this function will be executed when the program is run.

e.g. `func add(int i, int j) int { return i+j; }`

Standard Library

Array built-in functions

`lena(array)`
Returns the length (type int) of a given array.

e.g. `lena([1,2,3,4]); /* returns int of value 4 */`

`append(array, value)`
Appends value to the end of the array.

`arr_init()`
Initializes an empty array. Can then add elements to the array with the `append()` function.

`arr[index] = value`
Utilizes the index operator to change the value stored at `index` to `value`.

String built-in functions

`lens(string)`
Returns the length (type int) of a given array.

e.g. `lens("hello"); /* returns int of value 5 */`

Map built-in functions

`lenm(map)`
Returns the total number of key-value pairs in the map.

e.g. `lenm([1:2,3:4,5:6]); /* returns int of value 3 */`

`map[key] = value`
Utilizes the index operator to change the value stored at key `key` to `value`, or if `key` does not exist in the map, it adds the `key`, `value` pair to the map.

`getKeys(map)`
Returns an array of the keys from the map.

`map_init()`

Initializes an empty map. Can then add elements to the map using: `map[key] = value`

Edge built-in functions

`getSrc(edge)`
Returns source node struct.

`getDst(edge)`
Returns destination node struct.

`getVal(edge)`
Returns edge value struct.

`setSrc(edge, node_struct)`
Sets the source node of `edge` to `node_struct`.

`setDst(edge, node_struct)`
Sets the destination node of `edge` to `node_struct`.

`setVal(edge, node_struct)`
Sets the edge value of `edge` to `node_struct`.

Graph built-in functions

`graph_init()`
Initializes an empty graph. Edges can then be added to the graph using the `addEdge()` function.

`getNodes(graph)`
Returns an array of node structs.

`getEdges(graph, src_node)`
Returns an array of edges that have node `src_node` as source.

`addEdge(graph, new_edge)`
Adds edge `new_edge` to the graph.

Other built-in functions

`printi(value)`

Prints the contents of integer `value` to standard output.

`printd(value)`
Prints the contents of double `value` to standard output.

`prints(value)`
Prints the contents of string `value` to standard output.

`printc(value)`
Prints the contents of char `value` to standard output.

`printb(value)`
Prints the contents of boolean `value` to standard output (0 if false, 1 if true).

`int_of_double(double)`
Returns the value of a double as an int. The digits that follow the decimal point are truncated.

e.g. `int_of_double(5.3) /* returns 5 */`

`double_of_int(int)`
Returns the value of an int as a double.

e.g. `double_of_int(5) /* returns 5.0 */`

Project Plan

Our Process

During the ideation phase of our project, which was the writing of the LRM, we all pitched potential features that we should include in our language, discussed them, and reached consensus on what we should or shouldn't include in the language. The general theme of the programming language that we decided on was one that provide an interface for parsing and manipulating graphs.

Following the LRM, for each phase of the language that was due, i.e. scanner and parser, hello world, and project completion, we would scope out what needs to be completed and break it down into smaller chunks that can be distributed among the team equally. For example, the following is the task list that we used for scanner and parser:

Progress checklist for Scanner + Parser

- [x] Scanner
- [x] Parser
  - [x] typ
  - [x] fdecl
  - [x] stmt
    - [x] add if elseif else, and other control flow rules
  - [x] expr
    - [x] container data type literals
      - [x] struct
      - [x] array
      - [x] map
      - [x] graph

At the start of the project, we agreed upon a weekly time to meet, and towards the end of project completion we agreed upon a second weekly time so we could meet more often and move more quickly. Whenever we encountered any difficulties, we consulted our TA Dean for advice on implementation, such as on the use of LLVM and writing of the Makefile.

Programming Style Guide

From the get-go we did not agree on a formal style guide, but for the most part we followed the following conventions:

- Use proper indentation such that scope can be inferred
- Include comments that describes what a non-trivial piece of code does or doesn't do
- Ensure matching rules are fully exhaustive, e.g. with a catch-all wildcard rule that raises a failure message that also prints relevant context variables
- Use wildcards on unused variables
- Name constructors with TitleCase and functions and variables with snake_case

Project Timeline

- October 15: Finished scanner and parser
- November 11: Finished hello world
- November 25: Finished semantic checking for all language features, began writing end-to-end tests
- December 6: Finished all operations on base data types
- December 11: Finished arrays, maps, and structs
- December 15: Finished graphs
- December 18: Finished Dijkstra's algorithm implementation demo

Roles and Responsibilities

Member	Role	Work Done
Kevin Zeng	Manager/Language Guru	scanner, parser, ast, semant, sast, codegen, gae.c, testing, demo
Andrew Jones	System Architect	scanner, parser, ast, semant, sast, codegen, gae.c, testing
Samara Nebel	Tester	scanner, parser, ast, semant, sast, codegen, gae.c, testing

Software Development Environment

- GitHub was used to easily work on the same code as a group. For each addition that a member was working on, a branch was created. Then, when the addition was completed,

a pull request would be made so the others could take a look at the code and add comments or approve the merge into master.

- We made use of a Makefile for compiling the source code and creating an executable, and a Bash script for compiling GaE code
- Development environments differed for each group member, but the following were used extensively: docker, virtualbox, atom, sublime, vim

Project Log


```
930793f Kevin Zeng     Sun Dec 16 03:12:27 2018 +0000      shhh
174db2b Kevin Zeng     Sun Dec 16 03:08:13 2018 +0000      moved some files
b1ab655 Kevin Zeng     Sun Dec 16 02:49:53 2018 +0000      added some style to demo code
3b2b5af Kevin Zeng     Sun Dec 16 02:29:52 2018 +0000      removed all warnings!!!
9bc6f93 Kevin Zeng     Sun Dec 16 02:13:56 2018 +0000      forward decls to remove warnings
a4ea569 Andrew Jones   Tue Dec 18 17:21:05 2018 -0500      Script now just creates the
executable
721522d Kevin Zeng     Tue Dec 18 17:15:17 2018 -0500      Merge pull request #60 from
kzeng10/getnodes
8a2a4bd Kevin Zeng     Mon Dec 17 21:47:17 2018 -0500      Update README.md
c0038ab Kevin Zeng     Mon Dec 17 21:37:56 2018 -0500      Update README.md
e0e9710 Kevin Zeng     Mon Dec 17 11:40:23 2018 -0500      Merge pull request #59 from
kzeng10/fix-struct
5fc3fcb Kevin Zeng     Mon Dec 17 11:33:35 2018 -0500      Merge pull request #58 from
kzeng10/nodepair
301ab52 Kevin Zeng     Sun Dec 16 00:39:14 2018 -0500      Update README.md
954f53b Kevin Zeng     Sun Dec 16 00:36:19 2018 -0500      Merge pull request #56 from
kzeng10/getkeys
9b69eb8 Kevin Zeng     Sun Dec 16 00:36:09 2018 -0500      Merge pull request #54 from
kzeng10/demo
48f36f5 Kevin Zeng     Sat Dec 15 22:50:07 2018 -0500      Update README.md
4e06a2e Kevin Zeng     Sat Dec 15 20:56:10 2018 -0500      Merge pull request #55 from
kzeng10/struct-map
b84c4f4 Kevin Zeng     Sat Dec 15 20:55:53 2018 -0500      Update README.md
7e00fb0 Andrew Jones   Sat Dec 15 20:52:37 2018 -0500      Merge pull request #52 from
kzeng10/break/continue
232d01a Samara Rachel  Sat Dec 15 20:52:27 2018 -0500      Merge pull request #50 from
kzeng10/array_append
8b0da73 Samara Rachel  Sat Dec 15 20:52:03 2018 -0500      Merge branch 'master' into
array_append
```

```
0ddd115 Kevin Zeng     Sat Dec 15 20:30:23 2018 -0500      Merge pull request #53 from
kzeng10/graph
8b6a065 Kevin Zeng     Sat Dec 15 20:30:01 2018 -0500      Merge pull request #51 from
kzeng10/support-struct
c826232 Kevin Zeng     Sat Dec 15 17:56:15 2018 +0000      collatz conjecture
410490d Kevin Zeng     Sat Dec 15 17:15:05 2018 +0000      split it into a program that gives
length and a program that gives path
3e7ff5a Kevin Zeng     Sat Dec 15 16:55:14 2018 +0000      dijkstra works!!
289dfa7 Kevin Zeng     Sat Dec 15 15:10:11 2018 +0000      demo runs but doesn't give correct
answer, someone feel free to take over debugging i need to study for finals
9f9ae04 Kevin Zeng     Sat Dec 15 11:53:20 2018 +0000      Merge branch 'expose_inits' into
getnodes
9383ff0 Andrew Jones   Sun Dec 16 00:00:44 2018 -0500      Added boolean binops and median of
arrays demo
71d3cba Andrew Jones   Sat Dec 15 22:55:28 2018 -0500      Exposed map_init() arr_init() and
graph_init() functions
a926356 Kevin Zeng     Sat Dec 15 11:45:35 2018 +0000      finished in operator
4d14436 Kevin Zeng     Sat Dec 15 11:17:52 2018 +0000      added new test file for sanity, and
updated demo file so it is syntactically correct
7acae2b Kevin Zeng     Sat Dec 15 10:38:53 2018 +0000      fixed element checks to also
consider flags that we added
3e7fe6d Kevin Zeng     Sat Dec 15 10:09:01 2018 +0000      Merge branch 'master' into getnodes
71f8e5e Kevin Zeng     Sat Dec 15 10:07:27 2018 +0000      Merge branch 'master' into
fix-struct
9c0e051 Samara Rachel  Fri Dec 14 23:29:00 2018 -0500      Merge branch 'master' into
array_append
52b59db Samara Rachel  Fri Dec 14 23:22:41 2018 -0500      testing
87a501e Samara Rachel  Fri Dec 14 21:41:55 2018 -0500      testing
405222c Samara Rachel  Fri Dec 14 20:48:03 2018 -0500      error fixing
624c001 Samara Rachel  Fri Dec 14 20:40:42 2018 -0500      test
8e72a4e Andrew Jones   Fri Dec 14 20:32:05 2018 -0500      added test file for implementing
break/continue with if blocks
2da4029 Samara Rachel  Fri Dec 14 20:31:36 2018 -0500      error fixing
b256ee4 Samara Rachel  Fri Dec 14 20:30:11 2018 -0500      error fixing
0225655 Samara Rachel  Fri Dec 14 20:24:28 2018 -0500      error fixing
545a31c Samara Rachel  Fri Dec 14 20:22:49 2018 -0500      error fixing
7c9b52e Andrew Jones   Fri Dec 14 20:13:39 2018 -0500      Merge pull request #49 from
kzeng10/misc
25fc0bc Andrew Jones   Fri Dec 14 20:12:44 2018 -0500      Merge branch 'master' into misc
8df6693 Samara Rachel  Fri Dec 14 20:11:26 2018 -0500      added structs
229e655 Kevin Zeng     Sat Dec 15 01:06:51 2018 +0000      did it for map
c0e19bb Kevin Zeng     Sat Dec 15 01:00:47 2018 +0000      removed struct check for arrays by
using a contains_struct flag
8d894d6 Kevin Zeng     Sat Dec 15 00:47:25 2018 +0000      should be working
6029d54 Kevin Zeng     Fri Dec 14 19:44:01 2018 -0500      Update README.md
ffa6cf7 Kevin Zeng     Fri Dec 14 23:56:18 2018 +0000      organized test file a lil bit
75e87ad Kevin Zeng     Fri Dec 14 23:48:29 2018 +0000      Merge branch 'master' into nodepair
4edc6e4 Kevin Zeng     Fri Dec 14 23:40:36 2018 +0000      clean up a bit
f4f8dfd Kevin Zeng     Fri Dec 14 23:38:27 2018 +0000      getEdges works!
a05ed97 Kevin Zeng     Fri Dec 14 21:56:49 2018 +0000      nodepair works!
00cebdc Kevin Zeng     Fri Dec 14 20:35:38 2018 +0000      add new test case that involves for
loop
7a3b080 Kevin Zeng     Fri Dec 14 20:32:57 2018 +0000      getKeys
8c07307 Kevin Zeng     Fri Dec 14 20:01:29 2018 +0000      minor fixes
2f8e7ef Kevin Zeng     Fri Dec 14 19:49:05 2018 +0000      allow maps to have structs as keys
60ebc14 Kevin Zeng     Fri Dec 14 19:28:24 2018 +0000      updated demo file
4e5ead1 Kevin Zeng     Fri Dec 14 19:04:45 2018 +0000      Merge branch 'master' into graph
```

```
da53eff Kevin Zeng     Thu Dec 13 20:16:47 2018 +0000      expose graph add edge
b6657bb Kevin Zeng     Thu Dec 13 21:33:42 2018 +0000      array in for struct
30a7c37 Andrew Jones   Thu Dec 13 16:33:05 2018 -0500      Fixed get to set
9bb3a9d Samara Rachel  Thu Dec 13 16:26:41 2018 -0500      error fixing
48f3d73 Samara Rachel  Thu Dec 13 16:25:45 2018 -0500      error fixing
05bccbe Samara Rachel  Thu Dec 13 16:24:08 2018 -0500      error fixing
226b2d1 Samara Rachel  Thu Dec 13 16:23:11 2018 -0500      error fixing
f219167 Samara Rachel  Thu Dec 13 16:08:58 2018 -0500      error fixing
0524a54 Samara Rachel  Thu Dec 13 16:07:41 2018 -0500      error fixing
bf278ca Samara Rachel  Thu Dec 13 16:05:40 2018 -0500      exposed append
64386ae Andrew Jones   Thu Dec 13 15:38:38 2018 -0500      fixed some warnings, and added
int_of_double and double_of_int
6bbf427 Kevin Zeng     Thu Dec 13 15:21:17 2018 -0500      Update README.md
2e2a21d Kevin Zeng     Thu Dec 13 20:05:53 2018 +0000      graph init
6a302da Kevin Zeng     Thu Dec 13 14:57:51 2018 -0500      Merge pull request #48 from
kzeng10/struct-eq
417074f Kevin Zeng     Thu Dec 13 14:56:07 2018 -0500      Update README.md
f1f31e7 Samara Rachel  Thu Dec 13 14:54:23 2018 -0500      Update README.md
55c73a5 Andrew Jones   Thu Dec 13 14:49:05 2018 -0500      Added string length function
4f8f267 Kevin Zeng     Thu Dec 13 19:41:20 2018 +0000      Merge branch 'master' into graph
dcfe32b Andrew Jones   Thu Dec 13 14:41:05 2018 -0500      fixed unused variable warnings in
codegen
38304b7 Andrew Jones   Thu Dec 13 14:34:47 2018 -0500      fixed error in function names getsrc
getdst getval
fe769e5 Kevin Zeng     Thu Dec 13 19:33:51 2018 +0000      full impl
e9037f6 Andrew Jones   Thu Dec 13 14:30:58 2018 -0500      Update README.md
78b9923 Andrew Jones   Thu Dec 13 14:29:41 2018 -0500      increment and decrement working
893219e Kevin Zeng     Thu Dec 13 19:14:12 2018 +0000      c impl
e0d363c Kevin Zeng     Thu Dec 13 18:59:20 2018 +0000      wip
e297b6d Kevin Zeng     Thu Dec 13 13:55:37 2018 -0500      Update README.md
3d80008 Kevin Zeng     Thu Dec 13 13:54:35 2018 -0500      Merge pull request #47 from
kzeng10/edge
9bb1827 Samara Rachel  Thu Dec 13 13:54:12 2018 -0500      Merge pull request #46 from
kzeng10/stmt_if
cb0b4c2 Kevin Zeng     Thu Dec 13 13:54:06 2018 -0500      Update README.md
b41be82 Kevin Zeng     Thu Dec 13 13:48:26 2018 -0500      Merge pull request #45 from
kzeng10/semant-warnings
bbf2feb Samara Rachel  Thu Dec 13 13:46:45 2018 -0500      error fixing
0f81b6c Samara Rachel  Thu Dec 13 13:37:40 2018 -0500      error fixing
399e561 Samara Rachel  Thu Dec 13 13:36:50 2018 -0500      error fixing
855c40b Samara Rachel  Thu Dec 13 13:35:58 2018 -0500      error fixing
73e6ee6 Kevin Zeng     Thu Dec 13 13:35:54 2018 -0500      Merge pull request #44 from
kzeng10/map-in
6579a7c Kevin Zeng     Thu Dec 13 18:32:42 2018 +0000      Merge branch 'master' into edge
ffc42de Samara Rachel  Thu Dec 13 13:32:06 2018 -0500      error fixing
abd7f69 Kevin Zeng     Thu Dec 13 18:31:50 2018 +0000      Merge branch 'master' into edge
00bed7f Kevin Zeng     Thu Dec 13 18:31:02 2018 +0000      Merge branch 'master' into
semant-warnings
35c0044 Kevin Zeng     Thu Dec 13 18:30:26 2018 +0000      edge built-in functions codegen
1bab453 Samara Rachel  Thu Dec 13 13:27:31 2018 -0500      error fixing
aa1e94b Kevin Zeng     Thu Dec 13 18:20:50 2018 +0000      updated semant with edge built-ins
3581cf9 Samara Rachel  Thu Dec 13 13:20:37 2018 -0500      error fixing
bd39a60 Samara Rachel  Thu Dec 13 13:18:55 2018 -0500      error fixing
f736980 Samara Rachel  Thu Dec 13 13:17:28 2018 -0500      error fixing
1535c3d Andrew Jones   Thu Dec 13 13:15:02 2018 -0500      Update README.md
b6c99ce Samara Rachel  Wed Dec 12 17:42:08 2018 -0500      error fixing
25eddab Samara Rachel  Wed Dec 12 17:37:36 2018 -0500      error fixing
```

```
b060d0e Samara Rachel  Wed Dec 12 17:36:55 2018 -0500      error fixing
e491c83 Samara Rachel  Wed Dec 12 17:35:01 2018 -0500      error fixing
07145d4 Samara Rachel  Wed Dec 12 17:33:35 2018 -0500      error fixing
727ee31 Samara Rachel  Wed Dec 12 17:25:07 2018 -0500      error fixing
496609c Samara Rachel  Wed Dec 12 17:24:29 2018 -0500      error fixing
192169c Samara Rachel  Wed Dec 12 17:22:38 2018 -0500      error fixing
c77091e Samara Rachel  Wed Dec 12 17:14:49 2018 -0500      for
ff318e5 Andrew Jones   Wed Dec 12 17:12:15 2018 -0500      Added unops
b902cd5 Samara Rachel  Wed Dec 12 17:10:25 2018 -0500      error fixing
7a0432a Samara Rachel  Wed Dec 12 17:08:49 2018 -0500      error fixing
0d2c026 Samara Rachel  Wed Dec 12 17:05:30 2018 -0500      error fixing
11e3e98 Samara Rachel  Wed Dec 12 17:04:51 2018 -0500      error fixing
5d9cf6e Samara Rachel  Wed Dec 12 17:03:40 2018 -0500      while
89df249 Samara Rachel  Wed Dec 12 16:58:27 2018 -0500      error fixing
3ea8cf2 Samara Rachel  Wed Dec 12 16:57:40 2018 -0500      error fixing
5ec3ffa Samara Rachel  Wed Dec 12 16:56:50 2018 -0500      removed elseif
b78eb67 Kevin Zeng     Wed Dec 12 16:40:48 2018 -0500      Update README.md
9ea37ef Samara Rachel  Wed Dec 12 16:38:43 2018 -0500      error fixing
ddf1d3a Samara Rachel  Wed Dec 12 16:37:50 2018 -0500      error fixing
3f89e51 Kevin Zeng     Wed Dec 12 16:35:19 2018 -0500      Merge pull request #43 from
kzeng10/structcg
e7abbc6 Samara Rachel  Wed Dec 12 16:26:21 2018 -0500      error fixing
8e7b163 Samara Rachel  Wed Dec 12 16:17:33 2018 -0500      error fixing
8e6726c Samara Rachel  Wed Dec 12 16:16:09 2018 -0500      elseif and tests
c7d877d Andrew Jones   Wed Dec 12 16:15:03 2018 -0500      Added strcmp and updated test for
structaccess
81b1019 Samara Rachel  Wed Dec 12 16:07:00 2018 -0500      tests
7f6763f Samara Rachel  Wed Dec 12 16:03:52 2018 -0500      added else
53d433a Samara Rachel  Wed Dec 12 16:01:51 2018 -0500      rev
1cca80e Samara Rachel  Wed Dec 12 16:00:18 2018 -0500      rev
d153cd7 Samara Rachel  Wed Dec 12 15:54:07 2018 -0500      error fixing
86cbf38 Kevin Zeng     Wed Dec 12 15:52:11 2018 -0500      Update README.md
53fd3ea Kevin Zeng     Wed Dec 12 15:45:00 2018 -0500      Merge pull request #42 from
kzeng10/map-void
dbf471a Kevin Zeng     Wed Dec 12 15:44:48 2018 -0500      Merge pull request #41 from
kzeng10/in
c4de430 Samara Rachel  Wed Dec 12 15:44:20 2018 -0500      error fixing
c66edea Samara Rachel  Wed Dec 12 15:43:05 2018 -0500      error fixing
d7d0c74 Samara Rachel  Wed Dec 12 15:39:19 2018 -0500      error fixing
3325e83 Kevin Zeng     Wed Dec 12 20:34:44 2018 +0000      Merge branch 'map-in' into
semant-warnings
ca8c92c Samara Rachel  Wed Dec 12 15:30:04 2018 -0500      error fixing
a51faa1 Kevin Zeng     Wed Dec 12 20:29:35 2018 +0000      suppress most warnings
a735491 Samara Rachel  Wed Dec 12 15:23:34 2018 -0500      error fixing
8718db9 Andrew Jones   Wed Dec 12 15:05:38 2018 -0500      Structs working
3afd465 Kevin Zeng     Wed Dec 12 19:57:26 2018 +0000      Merge branch 'master' into map-in
c7827da Samara Rachel  Wed Dec 12 14:54:57 2018 -0500      error fixing
7a49e79 Kevin Zeng     Wed Dec 12 19:49:57 2018 +0000      map in
1d9d8db Samara Rachel  Wed Dec 12 14:48:34 2018 -0500      error fixing
272e359 Samara Rachel  Wed Dec 12 14:42:30 2018 -0500      error fixing
36db4a4 Samara Rachel  Wed Dec 12 14:40:32 2018 -0500      error fixing
ec040b4 Samara Rachel  Wed Dec 12 14:38:46 2018 -0500      error fixing
a77ad1a Samara Rachel  Wed Dec 12 14:24:13 2018 -0500      error fixing
8056308 Samara Rachel  Wed Dec 12 14:20:00 2018 -0500      error fixing
b138e16 Samara Rachel  Wed Dec 12 14:13:35 2018 -0500      error fixing
90b2f1d Samara Rachel  Wed Dec 12 14:12:39 2018 -0500      error fixing
b1bcf30 Samara Rachel  Wed Dec 12 14:09:40 2018 -0500      error fixing
```

```
9b7a8e8 Samara Rachel  Wed Dec 12 14:08:15 2018 -0500      error fixing
3992563 Kevin Zeng     Wed Dec 12 19:04:27 2018 +0000      Merge branch 'in' into map-in
e8bd22b Kevin Zeng     Wed Dec 12 18:58:07 2018 +0000      organize files a bit
774c5c6 Kevin Zeng     Wed Dec 12 18:57:00 2018 +0000      allow maps to have void * key
0a7c79a Samara Rachel  Wed Dec 12 13:52:02 2018 -0500      error fixing
137467f Kevin Zeng     Wed Dec 12 18:34:57 2018 +0000      allow maps to have void * value
05fcefe Samara Rachel  Wed Dec 12 13:33:41 2018 -0500      error fixing
566eb4d Samara Rachel  Wed Dec 12 13:30:08 2018 -0500      error fixing
3e3e75f Samara Rachel  Wed Dec 12 13:23:03 2018 -0500      error fixing
ff83834 Samara Rachel  Wed Dec 12 13:19:21 2018 -0500      if builder
b299829 Kevin Zeng     Wed Dec 12 18:14:32 2018 +0000      array in
1cf09a1 Kevin Zeng     Wed Dec 12 12:51:53 2018 -0500      Update README.md
c822266 Kevin Zeng     Wed Dec 12 12:49:33 2018 -0500      Update README.md
87dd737 Samara Rachel  Wed Dec 12 12:39:13 2018 -0500      Merge pull request #40 from
kzeng10/arr_index
0ea7985 Kevin Zeng     Wed Dec 12 17:01:07 2018 +0000      print rule for in
ae205a0 Samara Rachel  Tue Dec 11 17:49:27 2018 -0500      Merge branch 'master' into arr_index
39923cc Samara Rachel  Tue Dec 11 17:47:24 2018 -0500      error fixing
6fa7f26 Kevin Zeng     Tue Dec 11 17:35:25 2018 -0500      Merge pull request #39 from
kzeng10/len
ac671cf Kevin Zeng     Tue Dec 11 17:33:01 2018 -0500      Update README.md
8a75a9a Kevin Zeng     Tue Dec 11 17:32:07 2018 -0500      Merge pull request #38 from
kzeng10/map-assign
9187fa2 Samara Rachel  Tue Dec 11 17:31:09 2018 -0500      assign
fb51c4c Samara Rachel  Tue Dec 11 17:29:56 2018 -0500      index assign
4f62306 Andrew Jones   Tue Dec 11 17:07:38 2018 -0500      Merge branch 'master' of
https://github.com/kzeng10/gae into structcg
6f3008f Andrew Jones   Tue Dec 11 17:07:34 2018 -0500      working on structs
8f1c66f Kevin Zeng     Tue Dec 11 16:58:37 2018 -0500      Update README.md
68dbf12 Kevin Zeng     Tue Dec 11 16:56:09 2018 -0500      Merge pull request #36 from
kzeng10/assign
49763b3 Samara Rachel  Tue Dec 11 16:54:09 2018 -0500      Merge pull request #37 from
kzeng10/array_index
45bbfed Samara Rachel  Tue Dec 11 16:48:32 2018 -0500      rev
7a113de Samara Rachel  Tue Dec 11 16:41:35 2018 -0500      list.rev
347aee4 Samara Rachel  Tue Dec 11 16:38:28 2018 -0500      test
6d0f717 Samara Rachel  Tue Dec 11 16:37:18 2018 -0500      error fixing
4d2ff4d Samara Rachel  Tue Dec 11 16:33:58 2018 -0500      fixed sast
4cc9315 Samara Rachel  Tue Dec 11 16:29:35 2018 -0500      array index
89c0765 Samara Rachel  Tue Dec 11 16:12:45 2018 -0500      Merge pull request #35 from
kzeng10/cg_array
329b2f0 Samara Rachel  Tue Dec 11 16:12:13 2018 -0500      Update parser.mly
3510493 Samara Rachel  Tue Dec 11 16:06:08 2018 -0500      Merge branch 'master' into cg_array
58620be Kevin Zeng     Tue Dec 11 15:56:20 2018 -0500      Merge pull request #34 from
kzeng10/map
e6aefba Andrew Jones   Tue Dec 11 15:52:17 2018 -0500      Merge pull request #33 from
kzeng10/struct
0b254dd Samara Rachel  Tue Dec 11 15:51:00 2018 -0500      test
ec33f96 Kevin Zeng     Tue Dec 11 15:48:42 2018 -0500      Update README.md
4ee6271 Samara Rachel  Tue Dec 11 15:48:05 2018 -0500      error fixing
8173c80 Samara Rachel  Tue Dec 11 15:42:15 2018 -0500      test
4351b64 Samara Rachel  Tue Dec 11 15:40:50 2018 -0500      error fixing
73b6369 Andrew Jones   Tue Dec 11 15:23:17 2018 -0500      Added accessing values of struct
with '.' (i.e. var.value)
bd8d55f Samara Rachel  Tue Dec 11 14:41:05 2018 -0500      error fixing
7b196cc Samara Rachel  Tue Dec 11 14:39:24 2018 -0500      error fixing
9012a73 Samara Rachel  Tue Dec 11 14:34:52 2018 -0500      error fixing
```

```
e56a0f7 Samara Rachel  Tue Dec 11 14:31:57 2018 -0500      error fixing
6649bcf Samara Rachel  Tue Dec 11 14:22:59 2018 -0500      error fixing
bc917ab Samara Rachel  Tue Dec 11 14:22:02 2018 -0500      error fixing
37c7e7c Samara Rachel  Tue Dec 11 14:17:53 2018 -0500      error fixing
fbfd586 Samara Rachel  Tue Dec 11 14:15:57 2018 -0500      error fixing
389edc3 Samara Rachel  Tue Dec 11 14:10:58 2018 -0500      error fixing
886df32 Samara Rachel  Tue Dec 11 14:06:42 2018 -0500      error fixing
1f4ad67 Samara Rachel  Tue Dec 11 14:05:42 2018 -0500      fixing errors
eec45a8 Samara Rachel  Tue Dec 11 13:49:05 2018 -0500      error fixing
14c54c8 Samara Rachel  Tue Dec 11 13:45:54 2018 -0500      working on array and index
a0b64f4 Samara Rachel  Tue Dec 11 13:35:19 2018 -0500      fixed error
b50d90a Kevin Zeng     Tue Dec 11 13:35:02 2018 -0500      Merge pull request #32 from
kzeng10/index
0c92c70 Samara Rachel  Tue Dec 11 13:26:00 2018 -0500      fixed type error
e3259c7 Kevin Zeng     Tue Dec 11 00:58:20 2018 +0000      len
8aaa15e Kevin Zeng     Tue Dec 11 00:23:11 2018 +0000      Merge branch 'master' into
map-assign
073bd81 Kevin Zeng     Tue Dec 11 00:18:55 2018 +0000      map set
1fc9f05 Kevin Zeng     Mon Dec 10 23:58:54 2018 +0000      updated assign rule in codegen
58930eb Kevin Zeng     Mon Dec 10 23:46:51 2018 +0000      Merge branch 'master' into assign
50ab781 Kevin Zeng     Mon Dec 10 23:46:05 2018 +0000      update rules through semant to
support expr on left side of assign
a99c73d Kevin Zeng     Mon Dec 10 23:20:57 2018 +0000      Merge branch 'master' into map
85226f4 Kevin Zeng     Mon Dec 10 23:05:28 2018 +0000      split up map methods by type,
separated index rule into mapindex and arrayindex, implemented map get and init
87ec0a0 Kevin Zeng     Mon Dec 10 21:08:41 2018 +0000      Merge branch 'master' into map
9cc0b73 Samara Rachel  Thu Dec 6 21:18:19 2018 -0500arrays added
0eed1c6 Samara Rachel  Thu Dec 6 21:12:42 2018 -0500index added
84b43f8 Samara Rachel  Thu Dec 6 21:11:07 2018 -0500index added
941fa9e Andrew Jones   Thu Dec 6 20:19:36 2018 -0500Make clean now removes .ll files
59ac54b Andrew Jones   Thu Dec 6 20:14:16 2018 -0500Merge pull request #31 from
kzeng10/string_codegen
0fedb22 Andrew Jones   Thu Dec 6 20:13:38 2018 -0500Merge branch 'master' into string_codegen
43dc169 Andrew Jones   Thu Dec 6 20:11:31 2018 -0500Fixed compile.sh to work with tests in the
e2etests folder
cafef0c Andrew Jones   Thu Dec 6 19:51:37 2018 -0500String binops working, added string_of_op
for doubles
f4aeba6 Kevin Zeng     Thu Dec 6 23:28:20 2018 +0000blah
485c78e Kevin Zeng     Thu Dec 6 23:27:40 2018 +0000undo misc changes in semant
1673b0a Kevin Zeng     Thu Dec 6 23:25:27 2018 +0000update gitignore
88f21b2 Kevin Zeng     Thu Dec 6 23:21:02 2018 +0000implement index for parser, ast, sast,
semant
7fa2c3c Kevin Zeng     Thu Dec 6 22:09:16 2018 +0000working map_init
5524832 Kevin Zeng     Thu Dec 6 22:06:17 2018 +0000Merge branch 'master' into map
0599edc Kevin Zeng     Mon Dec 3 23:10:36 2018 +0000got things to work
ecd7164 Andrew Jones   Thu Dec 6 11:06:14 2018 -0500Fixed testall script to find llvm.bitreader
to work with updated codegen
ef526ef Andrew Jones   Mon Dec 3 22:48:02 2018 -0500Added string binops, added makefile and
compiling shell script
04a4a74 Kevin Zeng     Mon Dec 3 20:21:25 2018 -0500Update README.md
9c2506e Kevin Zeng     Mon Dec 3 22:27:32 2018 +0000compile map init codegen
fe6ad84 Kevin Zeng     Mon Dec 3 21:09:15 2018 +0000init map.c
7630ea4 Kevin Zeng     Wed Nov 28 23:13:44 2018 -0500      Update README.md
074be4b Kevin Zeng     Wed Nov 28 23:09:36 2018 -0500      Merge pull request #29 from
kzeng10/fix-error-msg
b898c22 Samara Rachel  Wed Nov 28 23:09:16 2018 -0500      Merge pull request #30 from
kzeng10/pretty_print
```

```
25ba16c Samara Rachel  Wed Nov 28 23:08:44 2018 -0500      added semis
6cde195 Samara Rachel  Wed Nov 28 23:05:10 2018 -0500      fixed doubles
de9ed1c Andrew Jones   Wed Nov 28 22:48:22 2018 -0500      Merge branch 'master' into
pretty_print
8ca9696 Andrew Jones   Wed Nov 28 22:39:44 2018 -0500      fixed up for and while
7b8ff10 Kevin Zeng     Thu Nov 29 03:35:39 2018 +0000      Merge branch 'master' into
fix-error-msg
19e2abe Kevin Zeng     Wed Nov 28 22:34:54 2018 -0500      Update README.md
96f6af6 Samara Rachel  Wed Nov 28 22:33:58 2018 -0500      reversed for and while, added tests
d1b2111 Andrew Jones   Wed Nov 28 22:25:46 2018 -0500      Merge pull request #28 from
kzeng10/string_of_expr
97ea67d Andrew Jones   Wed Nov 28 22:23:13 2018 -0500      Fixed comparisons in semant (<> and
=) and fixed reversed lists for arrays, structs, maps, and graphs
18b3c31 Andrew Jones   Wed Nov 28 22:06:53 2018 -0500      finished up ifelseifs
25e1e87 Kevin Zeng     Thu Nov 29 02:22:06 2018 +0000      made failure messages more helpful,
cleaned up helper functions
d286d6e Samara Rachel  Wed Nov 28 20:41:04 2018 -0500      added "s"
777fef0 Samara Rachel  Wed Nov 28 20:36:07 2018 -0500      added ifelseif to pretty print and
other small edits
5616ac8 Andrew Jones   Wed Nov 28 20:31:59 2018 -0500      Fixed issue in comparing duplicate
types, added edge test
38e59d3 Andrew Jones   Wed Nov 28 18:42:48 2018 -0500      Updated sast string_of_expr, issue
in semant with checking struct types are same in graphs
924366f Andrew Jones   Wed Nov 28 18:29:54 2018 -0500      added string_of_expr functions for
everything in ast.ml
9381e01 Kevin Zeng     Sun Nov 25 21:02:19 2018 -0500      Update README.md
b3268f1 Kevin Zeng     Mon Nov 26 01:46:57 2018 +0000      add struct printing in string of
program
13054e6 Kevin Zeng     Mon Nov 26 01:34:04 2018 +0000      Merge branch 'master' of
github.com:kzeng10/gae
4f431d1 Kevin Zeng     Mon Nov 26 01:33:21 2018 +0000      add pretty printing for struct
b84ba74 Kevin Zeng     Mon Nov 26 01:20:15 2018 +0000      add pretty printing for array
b1e8f98 Kevin Zeng     Sun Nov 25 20:12:32 2018 -0500      Update README.md
8275a5c Kevin Zeng     Sun Nov 25 20:11:56 2018 -0500      Merge pull request #27 from
kzeng10/graphlit
33e6210 Kevin Zeng     Mon Nov 19 05:43:59 2018 +0000      semant.ml: graph
04875c2 Kevin Zeng     Mon Nov 19 04:39:36 2018 +0000      wip
bd711be Kevin Zeng     Mon Nov 19 04:38:04 2018 +0000      semant.ml: edgelit
b8f3976 Kevin Zeng     Sun Nov 25 19:13:08 2018 -0500      Merge pull request #25 from
kzeng10/maplit
9de5a2f Kevin Zeng     Tue Nov 20 12:21:11 2018 -0500      Update README.md
8fbbd3f Kevin Zeng     Mon Nov 19 04:40:11 2018 +0000      fix check_all_diff
dca7ae4 Andrew Jones   Sun Nov 18 22:55:52 2018 -0500      Merge pull request #24 from
kzeng10/printfunc
ca881df Andrew Jones   Sun Nov 18 22:49:21 2018 -0500      added test script for printing
primitives
719350b Kevin Zeng     Mon Nov 19 03:27:14 2018 +0000      remove map_key type, maplit expr
30a8c28 Andrew Jones   Sun Nov 18 22:40:03 2018 -0500      fixed small error in printing
booleans
7774ba8 Kevin Zeng     Sun Nov 18 22:39:58 2018 -0500      Merge pull request #22 from
kzeng10/binop
a6d04ce Andrew Jones   Sun Nov 18 22:37:47 2018 -0500      Added print functions for int,
double, string, char, bool
fa574d1 Kevin Zeng     Mon Nov 19 02:48:32 2018 +0000      rm todo
cea4d50 Kevin Zeng     Mon Nov 19 02:48:14 2018 +0000      graph in binop
29fa712 Kevin Zeng     Mon Nov 19 02:36:35 2018 +0000      modify edge to be edge<node type,
edge value type>
```

```
aed69ad Kevin Zeng     Mon Nov 19 02:28:32 2018 +0000      semant.ml: in binop for map and
array
1bd019e Kevin Zeng     Mon Nov 19 02:11:33 2018 +0000      Add Edge data structure. Type
follows the syntax of `edge<Int>`, and an edge literal is just a three-tuple of structs
e43a41d Kevin Zeng     Sun Nov 18 21:10:35 2018 -0500      Update README.md
5992ac5 Kevin Zeng     Mon Nov 19 01:52:59 2018 +0000      implement binop for int/double in
semant.ml
50b3df4 Kevin Zeng     Sun Nov 18 20:45:51 2018 -0500      Update README.md
22d0514 Andrew Jones   Sun Nov 18 20:34:00 2018 -0500      Merge pull request #21 from
kzeng10/scannerfix
dbedf66 Andrew Jones   Sun Nov 18 20:33:15 2018 -0500      deleted gae.native
e91d4bd Kevin Zeng     Sun Nov 18 20:32:46 2018 -0500      Update README.md
3170d02 Andrew Jones   Sun Nov 18 20:30:52 2018 -0500      Fixed scanning struct field to not
include the colon
8c56a12 Kevin Zeng     Sun Nov 18 20:28:02 2018 -0500      updated demo.gae file to correct
comment syntax
ea7c352 Andrew Jones   Sun Nov 18 20:20:10 2018 -0500      Fixed scanning strings/chars to not
include quotes
88bea28 Kevin Zeng     Tue Nov 13 21:49:33 2018 -0500      Update README.md
f8e83c5 Kevin Zeng     Tue Nov 13 21:45:28 2018 -0500      remove semicolons from test.all
f1f6569 Kevin Zeng     Tue Nov 13 21:43:34 2018 -0500      add testall script
b776a79 Andrew Jones   Tue Nov 13 21:37:31 2018 -0500      HELLO WORLD!
448ae2e Samara Rachel  Tue Nov 13 21:12:54 2018 -0500      Merge pull request #20 from
kzeng10/array
2d61a72 Samara Rachel  Tue Nov 13 21:08:57 2018 -0500      Merge branch 'master' into array
59c8873 Samara Rachel  Sun Nov 11 21:34:32 2018 -0500      added array and reassign
7ed1f19 Samara Rachel  Sun Nov 11 21:25:56 2018 -0500      array and reassign added
c2384ca Andrew Jones   Sun Nov 11 20:55:58 2018 -0500      Updated files to generate llvm code,
printing 1 works, strings don't
3bc7964 Andrew Jones   Sun Nov 11 20:31:12 2018 -0500      Merge pull request #18 from
kzeng10/codegen
0d769fe Kevin Zeng     Sun Nov 11 20:22:12 2018 -0500      Merge pull request #19 from
kzeng10/semant
bc51f75 Kevin Zeng     Sun Nov 11 20:14:31 2018 -0500      got semant.ml to compile, still
definitely wip
397e0aa Kevin Zeng     Sun Nov 11 20:11:38 2018 -0500      semi-working sstruct
e74b60d Andrew Jones   Sun Nov 11 19:09:16 2018 -0500      Updated codegen.ml to fit our
language, added string type and commented out code to try and get hello world to work
3930f6a Kevin Zeng     Sun Nov 11 18:30:43 2018 -0500      add todos for semant.ml
d754be6 Kevin Zeng     Sun Nov 11 18:09:53 2018 -0500      add comment, checkpoint up until
check_function
4feaa71 Kevin Zeng     Thu Nov 1 18:36:57 2018 -0400find_struct
8eb044a Kevin Zeng     Thu Nov 1 17:47:19 2018 -0400check_struct
78f041c Kevin Zeng     Thu Nov 1 17:35:20 2018 -0400check_binds
5a6ff2a Kevin Zeng     Sun Nov 11 18:00:34 2018 -0500      add codegen.ml
21ada71 Kevin Zeng     Sun Nov 11 17:56:21 2018 -0500      add some comments
07cb909 Samara Rachel  Thu Nov 1 19:00:02 2018 -0400Merge pull request #17 from kzeng10/sast
8e8e979 Samara Rachel  Thu Nov 1 18:59:04 2018 -0400fixed typo
42ae497 Samara Rachel  Thu Nov 1 18:56:07 2018 -0400added true and false
40a1359 Kevin Zeng     Thu Nov 1 18:52:06 2018 -0400Update README.md
0d046dd Samara Rachel  Thu Nov 1 18:51:30 2018 -0400removed boollit, added dublit
58fc55c Samara Rachel  Thu Nov 1 18:37:24 2018 -0400sast builds native
65971b3 Kevin Zeng     Thu Nov 1 18:02:12 2018 -0400revise test script to just check on
accept/reject, prevent nested structs
1781128 Kevin Zeng     Thu Nov 1 17:30:39 2018 -0400init gae.ml, commenting out LLVM stuff, and
fix scanner so it actually works now
dbd59a4 Kevin Zeng     Thu Nov 1 17:17:23 2018 -0400add sast.ml and semant.ml from microc
```

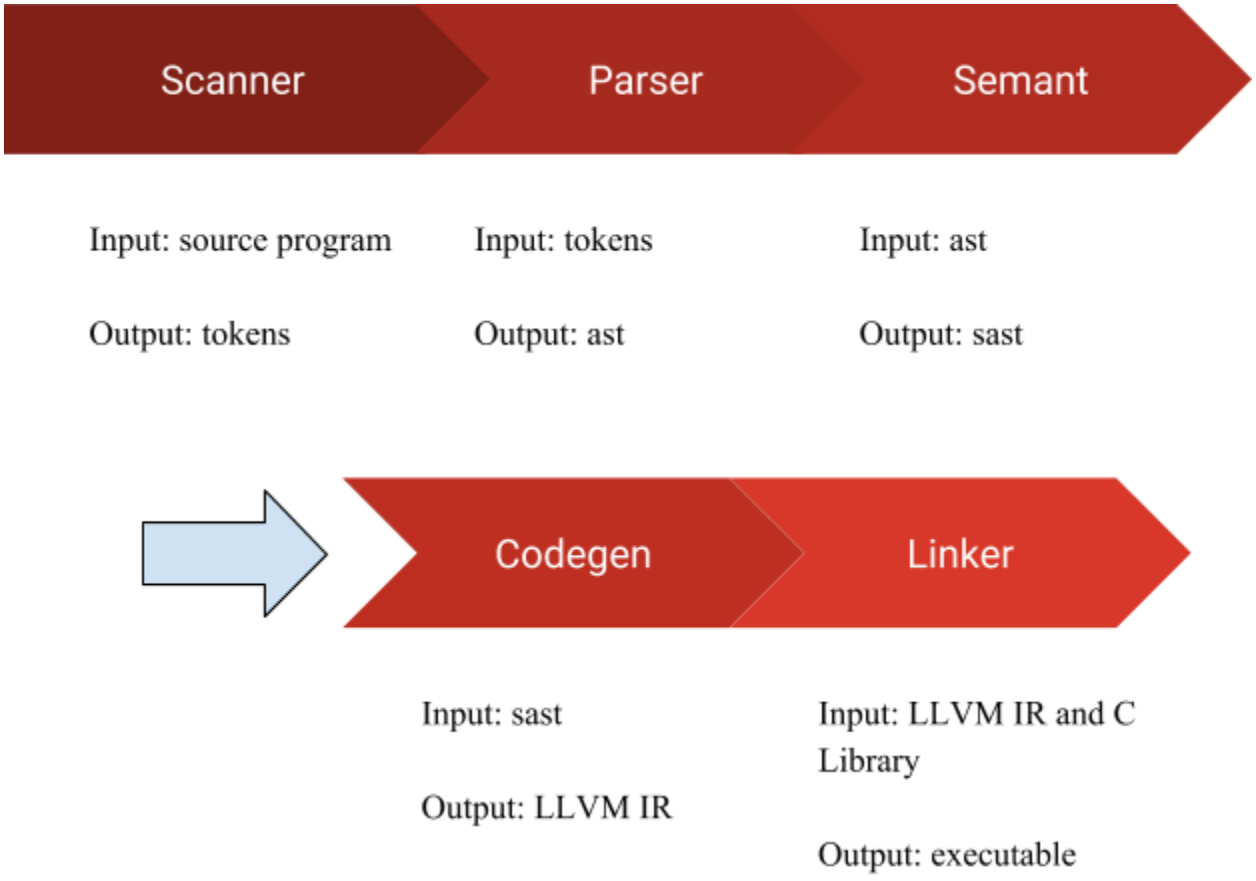
```
400f8f1 Samara Rachel  Mon Oct 29 22:55:52 2018 -0400      Merge pull request #16 from
kzeng10/if-statements
3d7d6cb Samara Rachel  Mon Oct 29 22:52:20 2018 -0400      Merge branch 'master' into
if-statements
c1e9bfe Kevin Zeng     Mon Oct 29 22:51:03 2018 -0400      Update README.md
575628c Samara Rachel  Mon Oct 29 22:49:31 2018 -0400      if, ifelse, ifelseif, ifelseifelse
statements
0f5fdc3 Andrew Jones   Mon Oct 29 22:35:27 2018 -0400      Merge pull request #15 from
kzeng10/vardeclassign
f917fef Kevin Zeng     Mon Oct 29 22:29:40 2018 -0400      add tests for fail test cases
2b81060 Andrew Jones   Mon Oct 29 22:29:04 2018 -0400      Added test cases for variable
declaration and assignment -- primitives, structs, arrays, maps, and graphs
df4b9d9 Kevin Zeng     Mon Oct 29 21:59:29 2018 -0400      update test script to include #
tests passed
ac95b2f Kevin Zeng     Mon Oct 29 21:55:43 2018 -0400      add func test
39ad8ef Kevin Zeng     Mon Oct 29 21:50:23 2018 -0400      fix struct lit rule, add struct test
a03abdd Kevin Zeng     Mon Oct 29 21:39:05 2018 -0400      add test for loops
daf486d Kevin Zeng     Mon Oct 29 21:31:59 2018 -0400      update test_ash.sh to allow newline
in tokens file, loops test
fec968a Kevin Zeng     Mon Oct 29 21:08:06 2018 -0400      fix types and ast
b627b9e Kevin Zeng     Mon Oct 29 20:41:22 2018 -0400      allow multiple statements in
if/for/while, restrict use of continue/break within for/while loops
761c17d Kevin Zeng     Mon Oct 29 20:25:28 2018 -0400      Merge pull request #14 from
kzeng10/init-test
e351ace Kevin Zeng     Mon Oct 29 20:25:04 2018 -0400      add base rule to stmt_list
27089c0 Kevin Zeng     Fri Oct 26 21:53:55 2018 -0400      add initial test_ast.sh test file to
test against menhir output
49065d8 Kevin Zeng     Fri Oct 26 21:37:35 2018 -0400      fixes some menhir warnings
c73ca1a Kevin Zeng     Fri Oct 26 21:13:46 2018 -0400      Merge pull request #12 from
kzeng10/ast
7f2024c Kevin Zeng     Fri Oct 26 21:11:48 2018 -0400      remove colon from struct_field rule
because FIELD already has a colon
3b69e42 Kevin Zeng     Fri Oct 26 20:44:49 2018 -0400      Merge branch 'ast' of
github.com:kzeng10/gae into ast
151dda1 Kevin Zeng     Fri Oct 26 20:44:42 2018 -0400      account for recursive elseif rule
833d8cc Samara Rachel  Thu Oct 25 18:18:15 2018 -0400      Merge branch 'master' into ast
bb0106e Samara Rachel  Thu Oct 25 18:03:06 2018 -0400      Merge pull request #11 from
kzeng10/Jason
62f0684 Kevin Zeng     Thu Oct 25 17:38:19 2018 -0400      Update README.md
b20b1cc Kevin Zeng     Thu Oct 25 17:36:54 2018 -0400      parser.native compiles properly
6ba2415 Kevin Zeng     Thu Oct 25 17:09:28 2018 -0400      Update README.md
155c8d6 Kevin Zeng     Thu Oct 25 17:05:23 2018 -0400      Update README.md
5feb20f Kevin Zeng     Thu Oct 25 16:17:37 2018 -0400      AST: typ, struct_decl, helper fns
6f2b3b7 Jason-Delancey Thu Oct 25 16:06:33 2018 -0400      added variables to scanner, token to
parser, and tried to make vdecl_list optional
b81e90b Kevin Zeng     Tue Oct 23 15:25:56 2018 -0400      Added demo file
26cc57c Kevin Zeng     Mon Oct 15 23:07:42 2018 -0400      Add incr/decr ops and some other
comments
87c363e Kevin Zeng     Mon Oct 15 22:57:05 2018 -0400      Merge branch 'master' of
github.com:kzeng10/gae
2790654 Kevin Zeng     Mon Oct 15 22:56:39 2018 -0400      Add rest of operations and a few
TODOs
0baa282 Kevin Zeng     Mon Oct 15 22:56:39 2018 -0400      Add rest of operations and a few
TODOs
b3d8abf Kevin Zeng     Mon Oct 15 22:19:46 2018 -0400      Update README.md
d4a3bb6 Kevin Zeng     Mon Oct 15 22:19:10 2018 -0400      Merge pull request #10 from
kzeng10/Jason
```

```
fe58e86 Kevin Zeng     Mon Oct 15 22:18:41 2018 -0400      Merge branch 'Jason' of
github.com:kzeng10/gae into Jason
36d27b5 Kevin Zeng     Mon Oct 15 22:17:57 2018 -0400      effect gitignore
f48bd54 Kevin Zeng     Mon Oct 15 22:17:57 2018 -0400      effect gitignore
428e9e6 Kevin Zeng     Mon Oct 15 22:16:46 2018 -0400      Fix merge conflicts
67eb1b8 Kevin Zeng     Mon Oct 15 22:15:23 2018 -0400      Merge branch 'Jason' of
github.com:kzeng10/gae into Jason
5e1905d Jason-Delancey Mon Oct 15 22:09:34 2018 -0400      updated array declaration
eac8684 Jason-Delancey Mon Oct 15 22:01:04 2018 -0400      updated array
af3f10b Jason-Delancey Mon Oct 15 21:59:23 2018 -0400      updated array for parser
fd8f208 Jason-Delancey Mon Oct 15 21:38:23 2018 -0400      updated array for parser
312bf0f Jason-Delancey Mon Oct 15 21:25:25 2018 -0400      added array to parser
ee67e3f Jason-Delancey Mon Oct 15 22:09:34 2018 -0400      updated array declaration
2bca71e Jason-Delancey Mon Oct 15 22:05:54 2018 -0400      updated array
6abac75 Jason-Delancey Mon Oct 15 22:01:04 2018 -0400      updated array
1c50817 Jason-Delancey Mon Oct 15 21:59:23 2018 -0400      updated array for parser
d5c4a2c Kevin Zeng     Mon Oct 15 21:52:54 2018 -0400      Update README.md
5f2769f Kevin Zeng     Mon Oct 15 21:52:38 2018 -0400      Merge pull request #4 from
kzeng10/samara
c04cd54 Kevin Zeng     Mon Oct 15 21:50:44 2018 -0400      Merge branch 'samara' of
github.com:kzeng10/gae into samara
170ee28 Kevin Zeng     Mon Oct 15 21:49:44 2018 -0400      Rebase onto master
7f919a8 Samara Rachel  Mon Oct 15 21:41:03 2018 -0400      added todo
0426a18 Samara Rachel  Mon Oct 15 21:35:15 2018 -0400      added just if elseif
604fe9e Samara Rachel  Mon Oct 15 21:30:44 2018 -0400      made if elseif else recursive
5e1bba1 Samara Rachel  Mon Oct 15 20:02:05 2018 -0400      added if elseif else
2cde1d5 Samara Rachel  Mon Oct 15 19:51:05 2018 -0400      added if elseif else
fd4876e Samara Rachel  Mon Oct 15 21:41:03 2018 -0400      added todo
b60cba6 Jason-Delancey Mon Oct 15 21:38:23 2018 -0400      updated array for parser
082b543 Samara Rachel  Mon Oct 15 21:35:15 2018 -0400      added just if elseif
a4587b0 Samara Rachel  Mon Oct 15 21:30:44 2018 -0400      made if elseif else recursive
471ba9d Jason-Delancey Mon Oct 15 21:25:25 2018 -0400      added array to parser
8a6d71f Kevin Zeng     Mon Oct 15 21:21:01 2018 -0400      Update README.md
b9bcce1 Kevin Zeng     Mon Oct 15 21:19:46 2018 -0400      Add some TODOs, wrapped container
structs with function
3d59d1a Andrew Jones   Mon Oct 15 21:16:17 2018 -0400      Merge pull request #9 from
kzeng10/graph
fa15ccf Kevin Zeng     Mon Oct 15 21:15:27 2018 -0400      Add graph decl rule
c08c3f1 Kevin Zeng     Mon Oct 15 21:14:13 2018 -0400      Add graph literal rules
555a69b Andrew Jones   Mon Oct 15 20:56:15 2018 -0400      Merge pull request #8 from
kzeng10/map
99222c2 Kevin Zeng     Mon Oct 15 20:55:19 2018 -0400      Add map decl rule
0caead0 Kevin Zeng     Mon Oct 15 20:50:07 2018 -0400      Add map literal rules
64362b6 Kevin Zeng     Mon Oct 15 20:47:01 2018 -0400      Update README.md
f1fb57e Andrew Jones   Mon Oct 15 20:46:23 2018 -0400      Merge pull request #7 from
kzeng10/struct
1ef8021 Kevin Zeng     Mon Oct 15 20:44:51 2018 -0400      Add struct definition rules
54fa137 Kevin Zeng     Mon Oct 15 20:13:25 2018 -0400      Add struct literal rule
fb97f71 Andrew Jones   Mon Oct 15 20:23:50 2018 -0400      Merge pull request #5 from
kzeng10/stringlit
6612684 Kevin Zeng     Mon Oct 15 20:22:34 2018 -0400      Add rules for literals
5ac29fe Kevin Zeng     Mon Oct 15 20:07:00 2018 -0400      Merge pull request #2 from
kzeng10/Jason-Delancey-patch-2
7046cb2 Kevin Zeng     Mon Oct 15 20:06:51 2018 -0400      Merge pull request #1 from
kzeng10/Jason-Delancey-patch-1
1012e4c Samara Rachel  Mon Oct 15 20:02:05 2018 -0400      added if elseif else
b8e76d5 Andrew Jones   Mon Oct 15 19:56:54 2018 -0400      Merge https://github.com/kzeng10/gae
```

```
5b9f4eb Andrew Jones   Mon Oct 15 19:55:42 2018 -0400      Added double literal, changed name
of int literal, updated char literal.
8a9ec5d Samara Rachel  Mon Oct 15 19:51:05 2018 -0400      added if elseif else
cc84c34 Samara Rachel  Mon Oct 15 19:44:29 2018 -0400      Merge pull request #3 from
kzeng10/master
5b57c38 Jason Delancey Mon Oct 15 19:43:00 2018 -0400      Update scanner.mll
790b7b5 Kevin Zeng     Mon Oct 15 19:37:10 2018 -0400      Update README.md
7de6d89 Jason Delancey Mon Oct 15 19:32:08 2018 -0400      Update parser.mly
5173c09 Kevin Zeng     Mon Oct 15 19:26:17 2018 -0400      Update README.md
c0a6f1f Samara Rachel  Mon Oct 15 19:18:00 2018 -0400      Merge branch 'master' of
https://github.com/kzeng10/gae
975aa5b Samara Rachel  Mon Oct 15 19:17:23 2018 -0400      added types
0a0e0ed Andrew Jones   Mon Oct 15 19:12:51 2018 -0400      Removed typdef, added map and graph
keywords.
379a2ce Andrew Jones   Sun Oct 14 21:31:31 2018 -0400      Merge https://github.com/kzeng10/gae
0aee97f Andrew Jones   Sun Oct 14 21:30:29 2018 -0400      Updated scanner for our language.
47f5a14 Kevin Zeng     Sun Oct 14 21:17:33 2018 -0400      Merge branch 'master' of
github.com:kzeng10/gae
794b6fe Kevin Zeng     Sun Oct 14 21:15:09 2018 -0400      Add if/while/for statements and
TODOs
5d344e3 Jason Delancey Sun Oct 14 21:05:23 2018 -0400      Update ast.ml
393a331 Jason Delancey Sun Oct 14 21:00:47 2018 -0400      Update ast.ml
b487043 Kevin Zeng     Sun Oct 14 20:38:24 2018 -0400      Add tokens and rule assoc to parser
17239b9 Kevin Zeng     Fri Oct 12 22:39:24 2018 -0400      Add MicroC parsing files
a447e3c Kevin Zeng     Fri Oct 12 22:34:08 2018 -0400      Initial commit
```

Architectural Design

Diagram



The Interfaces

source.gae

This is the GaE program that needs to be compiled.

scanner.mll

Reads the source GaE code and does the lexical analysis, creating tokens from the source code text.

parser.mly

Reads the tokens from the scanner module, and checks to make sure it is syntactically correct. If there are no parsing errors, it will generate the abstract syntax tree.

ast.ml

The abstract syntax tree representation of our GaE source code.

semant.ml

This semantically checks our AST to make sure that all the types are correct and outputs the semantically checked abstract syntax tree.

sast.ml

This is the semantically checked abstract syntax tree representation of our GaE source code.

codegen.ml

This takes the SAST as input and generates the LLVM code.

gae.c

This is our C library where we have our implementations of our data types as well as any built in functions. This is linked to the LLVM code generated by codegen.ml.

gae.ml

This is the top-level of our compiler. It goes through scanner, parser, semant, and codegen. If no flags are specified when running this, it will generate and output the LLVM IR. If you pass the "-a" flag it will print out a representation of the AST and the "-s" flag will print out a representation of the SAST.

Implementers and Attribution

All of these files were worked on more or less evenly by the whole group.

Test Plan

Test Suites

Our test suites are folders named ast_tests and e2e_tests. For each new feature of our language, we added an accompanying test case to each folder.

Test Automation

To run ast_tests we wrote a shell script that runs all of the tests through Menhir, compares the output of the test to our expected output, and counts the number of successes and the number of failures.

To run e2e_tests we wrote a shell script to compile any given test case. The script automatically compiles and executes the given test.

Example Programs

You can find all of the code for our test cases in the Appendix.

Lessons Learned

Andrew Jones

I have learned quite a lot throughout this project. At the start, I didn't even know OCaml was a programming language and at the end, I at least knew enough to write some decent code. The most important lesson I learned was the importance of communication and just generally working well as a group. Many times throughout, I would bounce ideas off my group members and together we would come up with the best plan of action. Also, just having a second set of eyes is extremely helpful. If you think your code should work but it doesn't, have someone else take a look at it. They might find some simple error that you missed just because they haven't been staring at the same code for an hour. My advice to future teams would be to start early and set weekly meetings where you sit down and work together. This will create a rhythm and allow you to be significantly more productive than just working individually.

Samara Nebel

Coming into this project I knew that I was going to have a pretty big learning curve. In terms of my coding knowledge, I am leagues behind most of the students in the CS department. So, I was very nervous about the amount of work that I would actually be able to contribute to a project like this. But, with the help of my other group members I was able to contribute more than I thought I could and I learned a lot about programming along the way. I know that I could not have done this without their help, so I think the most important thing that I learned and the most valuable advice that I can impart is that even if you feel like you are the weak link of the group, do not let that discourage you from asking for help and contributing.

Kevin Zeng

Working on this project was a really big learning experience for me. Although I played around with functional programming languages in the past, I never used it in a project. Working with

OCaml was initially difficult and unfamiliar, but as the semester progressed I started to really enjoy it. However, I think the most important thing I learned was how to break up relatively large coding tasks into smaller ones, determine dependencies among them, and figure out how to assign them to people so everyone is contributing equally. There were certainly hiccups along the way, such as people being blocked occasionally by someone else's tasks, but this led us to do a fair bit of pair programming and that was also an interesting experience. My advice to future teams is to make sure that everyone is on the same page with regards to the state of the code base, such as what has already been implemented, how it was implemented, and what hasn't been done. There were times when there was a gap in understanding about the state of the codebase, leading to a dip in productivity during our meetings because time would be spent explaining the code instead of writing it. A good process we adopted was pull requests and merges, which I recommend every team should learn and use.

Appendix

---

e2e_tests

array.gae

```
struct Int {
  value: int
}

func main() int {
  int i;
  Int x;
  int[] var;
  string[] var2;
  Int[] var3;

  var := [5,4];
  var2 := ["hello", "world"];
  var3 := [{value: 1}, {value: 2}];

  printi(var[0]);
  var[0] = 3;
  printi(var[0]);

  append(var, 6);
  for i = 0; i < lena(var) ; i = i + 1{
      printi(var[i]);
  }

  append(var2, "hi");
  for i = 0; i < lena(var2) ; i = i + 1{
      prints(var2[i]);
  }

  append(var3, {value: 3});
```

```
  for i = 0; i < lena(var3) ; i = i + 1{
      x = var3[i];
      printi(x.value);
  }
}
```

assign.gae

```
func main() int{
  int x;
  x := 5;
  printi(x);
  x = 6;
  printi(x);
  return 0;
}
```

break_continue.gae

```
func main() int{
     int i;
     int j;

     for i = 0; i<10; i++{
          if (i==3){
               /*continue*/
               /*need to add at least one statement here or it will
be a parsing error*/
               i = 3;
          }
          else{
               if (i==8){
                    /*break*/
                    i = 10;
               }
               else{
                    printi(i);
               }
          }
     }
```

```
}
```

double_tests.gae

```
func main() int {
  double var;
  var := 1.1;
  var = var +. 2.1;
  var = var /. 2.0;
  printd(var);
  var = var -. 0.54;
  var = var *. 1.12;
  printd(var);
}
```

edge.gae

```
struct Node {
  value: int
}

struct EdgeVal {
  value: string
}

func main() int {
  edge<Node, EdgeVal> var;
  Node src;
  Node dst;
  Node foo1;
  EdgeVal val;
  EdgeVal foo2;
  dst := {value: 1};
  val := {value: "hello world"};
  var := ({value: 3}, dst, val);
  src = getSrc(var);
  src.value = 5;
  setSrc(var, src);
  foo1 := getDst(var);
  printi(foo1.value);
```

```
  src.value = 10;
  /* Should this be the same as above? i.e., should we be passing
structs by ref or by value? */
  foo1 = getSrc(var);
  printi(foo1.value);
  foo2 = getVal(var);
  prints(foo2.value);
  return 0;
}
```

for.gae

```
func main() int{
  int i;
  int x;
  x := 0;

  for i = 10; i > 0; i--
  {
      x = x + i;
      printi(x);
  }
}
```

graph.gae

```
struct Int {
  value: int
}

func main() int {
  graph<Int, Int> var;
  edge<Int, Int> e;
  edge<Int, Int>[] edges;
  Int node;
  var := { ({value: 1}, {value: 2}, {value: 30}), ({value: 3},
{value: 3}, {value: 6}), ({value: 3}, {value: 4}, {value: 6}) };
  node := {value: 3};
  e := (node, node, node);
```

```
  /* addEdge/getEdges */
  prints("testing addEdge and getEdges");
  addEdge(var, e);
  edges := getEdges(var);
  e = edges[0];
  node = getVal(e);
  printi(node.value);
  e = edges[1];
  node = getVal(e);
  printi(node.value);
  e = edges[2];
  node = getVal(e);
  printi(node.value);
  return 0;
}
```

if.gae

```
func main() int{
  int x;
  x := 6;
  if x == 6
  {
    x = x - 2;
    x = x - 1;
    printi(x);
  }
  return 0;
}
```

ifelse.gae

```
func main() int{
  int x;
  x := 5;
  if x == 6
  {
    printi(x);
  }
  else
```

```
  {
    x = x - 2;
    x = x - 1;
    printi(x);
  }
  return 0;
}
```

in.gae

```
struct Int {
  value: int
}

func main() int {
  int[] var;
  string[] var2;
  Int[] var3;
  map<int, int> map1;
  map<string, int> map2;
  var := [5,4];
  var2 := ["hello", "world"];
  var3 := [{value: 1}, {value: 2}, {value: 3}];
  map1 := [1:1, 2:2];
  map2 := ["foo":2, "bar":5];

  printb(5 in var);
  printb(3 in var);
  printb("hello" in var2);
  printb("not here" in var2);
  printb({value: 1} in var3);
  printb({value: 5} in var3);
  printb(1 in map1);
  printb(5 in map1);
  printb("foo" in map2);
  printb("not here" in map2);
}
```

int-double.gae

```
func main() int {
```

```
    int x;
    int xx;
    double y;
    double yy;
    x := 3;
    y := 3.7;
    xx = int_of_double(y);
    yy = double_of_int(x);
    printi(xx);
    yy = yy +. 1.1;
    printd(yy);
}
```

len.gae

```
func main() int {
  map<string, int> m;
  int[] a;
  a := [1,2,3];
  m := ["one":1, "two":2];
  printi(lena(a));
  printi(lenm(m));
  return 0;
}
```

map_get_keys.gae

```
struct Int {
  value: int
}

func main() int {
  map<string, int> var;
  map<int, int> var2;
  map<Int, int> var4;
  string[] var5;
  int[] var6;
  Int[] var7;
  Int var8;
  int i;
  var := ["one":1, "two":2];
```

```
  var5 := getKeys(var);
  prints(var5[0]);
  var2 := [1:5, 5:3];
  var6 := getKeys(var2);
  printi(var6[0]);
  prints("keys are returned in reverse? lol");
  var4 := [{value: 3}: 3, {value: 5}: 5, {value: 6}: 5, {value: 10}:
5];
  var7 := getKeys(var4);
  for i := 0; i < lena(var7); i++ {
    var8 = var7[i];
    printi(var8.value);
  }
  prints("testing length of getKeys");
  printi(lena(getKeys(var4)));
  var4[{value: 3}] = 2;
  printi(lena(getKeys(var4)));
  var4[{value: 1}] = 1;
  printi(lena(getKeys(var4)));
  return 0;
}
```

map.gae

```
struct Int {
  value: int
}

func main() int {
  map<string, int> var;
  map<int, int> var2;
  map<string, string> var3;
  map<Int, int> var4;
  Int var5;
  map<Int, Int> var6;
  var := ["one":1, "two":2];
  printi(var["one"]);
  var["one"] = 11;
  printi(var["one"]);
  var2 := [1:5, 5:3];
```

```
  printi(var2[5]);
  var2[5] = 33;
  printi(var2[5]);
  var3 := ["hello":"world", "foo":"bar"];
  prints(var3["foo"]);
  var3["foo"] = "baz";
  prints(var3["foo"]);
  var5 := {value: 5};
  var4 := [{value: 3}: 3, {value: 5}: 5];
  printi(var4[var5]);
  var4[var5] = 10;
  printi(var4[var5]);
  var6 := [{value: 5}: {value: 11}];
  var5 = var6[var5];
  printi(var5.value);
  return 0;
}
```

map_init.gae

```
struct Int {
  value: int
}

func main() int {
  map<string, int> var;
  map<int, int> var2;
  map<string, string> var3;
  map<Int, int> var4;
  Int var5;
  var := map_init();
  /*var = ["one":1, "two":2];*/
  var["one"] = 1;
  var["two"] = 2;
  printi(var["one"]);
  var["one"] = 11;
  printi(var["one"]);
  var2 := [1:5, 5:3];
  printi(var2[5]);
  var2[5] = 33;
```

```
  printi(var2[5]);
  var3 := ["hello":"world", "foo":"bar"];
  prints(var3["foo"]);
  var3["foo"] = "baz";
  prints(var3["foo"]);
  var5 := {value: 5};
  var4 := [{value: 3}: 3, {value: 5}: 5];
  printi(var4[var5]);
  var4[var5] = 10;
  printi(var4[var5]);
  return 0;
}
```

misc.gae

```
struct Int {
  value: int
}

func main() int {
  graph<Int, Int> g;
  map<Int, bool> visited;
  Int[] nodes;
  Int node;
  int i;
  g = {
    ({value: 1},{value: 2},{value: 1}), /* syntax: (src, dest, edge)
*/
    ({value: 1},{value: 3},{value: 1}),
    ({value: 1},{value: 4},{value: 1}),
    ({value: 2},{value: 3},{value: 3}),
    ({value: 2},{value: 4},{value: 5}),
    ({value: 3},{value: 4},{value: 1})
  };
  nodes := getNodes(g);
  visited := map_init();
  for i := 0; i < lena(nodes); i++ {
    node := nodes[i];
    visited[node] = false;
    prints("---");
```

```
    printi(lena(getKeys(visited)));
    printi(node.value);
    printb(visited[node]);
    prints("===");
  }
  printi(lena(getKeys(visited)));

  return 0;
}
```

stringtests.gae

```
func main() int {
  string var;
  int length;
  var := "hello";
  length := lens(var);
  printi(length);
  var = var + " world";
  prints(var);
  prints("Testing string equals - should print 1:");
  printb(var == "hello world");
  prints("Testing string not equals - should print 1:");
  printb(var != "hello");
}
```

struct.gae

```
struct Int {
  value: int
}

struct Int2 {
  value: int,
  value2: int
}

func main() int {
  Int foo;
  Int bar;
  foo := {value: 5};
```

```
  bar := {value: 5};
  printb(foo == bar);
  return 0;
}
```

structaccess.gae

```
struct Int {
  value: int,
  name: string
}

struct String {
  value: string
}

func main() int {
  Int x;
  String y;
  x := {value: 5, name: "name"};
  y := {value: "value"};
  printi(x.value);
  prints(x.name);
  prints(y.value);
  x.value = 6;
  x.name = "new name";
  y.value = "new value";
  printi(x.value);
  prints(x.name);
  prints(y.value);
  return 0;
}
```

while.gae

```
func main() int{
    int x;
    x := 0;

    while x != 10
    {
```

```
        x = x + 1;
        printi(x);
    }
    return 0;
}
```

test_print_primitives.gae

```
func main() int{
     printi(8);
     printd(1.2);
     printb(true);
     printc('a');
     prints("string");
}
```

ast_tests

fail_break_outside_loop.tokens

```
FUNC ID LPAREN RPAREN INT LBRACE
BREAK SEMI
RETURN SEMI
RBRACE
```

fail_continue_outside_loop.tokens

```
FUNC ID LPAREN RPAREN INT LBRACE
CONTINUE SEMI
RETURN SEMI
RBRACE
```

fail_graph_with_primitive.tokens

```
FUNC ID LPAREN RPAREN INT LBRACE
GRAPH LT INT COMMA INT GT ID SEMI
RBRACE
```

fail_map_of_array.tokens

```
FUNC ID LPAREN RPAREN INT LBRACE
MAP LT INT LBRACKET RBRACKET COMMA INT GT ID SEMI
RETURN SEMI
RBRACE
```

fail_nested_array.tokens

```
FUNC ID LPAREN RPAREN INT LBRACE
INT LBRACKET RBRACKET LBRACKET RBRACKET ID SEMI
RETURN SEMI
RBRACE
```

test_array_decl_assign.tokens

```
FUNC ID LPAREN RPAREN INT LBRACE INT LBRACKET RBRACKET ID SEMI ID
ASSIGN LBRACKET INTLIT COMMA INTLIT COMMA INTLIT RBRACKET SEMI RBRACE
```

test_func.tokens

```
FUNC ID LPAREN INT ID COMMA STRING ID RPAREN INT LBRACE
STRUCTID ID SEMI
ID ASSIGN LBRACE FIELD INTLIT COMMA FIELD INTLIT RBRACE SEMI
RETURN INTLIT SEMI
RBRACE
FUNC ID LPAREN RPAREN STRUCTID LBRACE
ID LPAREN INTLIT COMMA STRLIT RPAREN SEMI
RETURN LBRACE FIELD INTLIT COMMA FIELD INTLIT RBRACE SEMI
RBRACE
FUNC ID LPAREN RPAREN STRING LBRACE
ID LPAREN RPAREN SEMI
RETURN STRLIT SEMI
RBRACE
```

test_graph_decl_assign.tokens

```
FUNC ID LPAREN RPAREN INT LBRACE GRAPH LT STRUCTID COMMA STRUCTID GT
ID SEMI ID ASSIGN LBRACE LPAREN LBRACE FIELD INTLIT RBRACE COMMA
LBRACE FIELD INTLIT RBRACE COMMA LBRACE FIELD INTLIT RBRACE RPAREN
COMMA LPAREN LBRACE FIELD INTLIT RBRACE COMMA LBRACE FIELD INTLIT
RBRACE COMMA LBRACE FIELD INTLIT RBRACE RPAREN RBRACE SEMI RBRACE
```

test_ifelseifelsestatments.tokens

```
FUNC ID LPAREN RPAREN INT
LBRACE

IF ID EQ ID
LBRACE
ID MOD ID SEMI
ID DIVIDE ID SEMI
RETURN ID SEMI
RBRACE
ELSEIF ID LT ID
LBRACE
ID TIMES ID SEMI
RETURN ID SEMI
RBRACE
```

```
ELSEIF ID GT ID
LBRACE
ID MINUS ID SEMI
RETURN ID SEMI
RBRACE
ELSE
LBRACE
ID MINUS ID SEMI
RETURN ID SEMI
RBRACE

RBRACE
```

test_ifelseifstatements.tokens

```
FUNC ID LPAREN RPAREN INT
LBRACE

IF ID EQ ID
LBRACE
ID MOD ID SEMI
ID DIVIDE ID SEMI
RETURN ID SEMI
RBRACE
ELSEIF ID LT ID
LBRACE
ID TIMES ID SEMI
RETURN ID SEMI
RBRACE
ELSEIF ID GT ID
LBRACE
ID MINUS ID SEMI
RETURN ID SEMI
RBRACE

RBRACE
```

test_ifelsestatments.tokens

```
FUNC ID LPAREN RPAREN INT
LBRACE

IF ID EQ ID
LBRACE
ID MOD ID SEMI
ID DIVIDE ID SEMI
RETURN ID SEMI
RBRACE
ELSE
LBRACE
ID MINUS ID SEMI
RETURN ID SEMI
RBRACE

RBRACE
```

test_ifstatements.tokens

```
FUNC ID LPAREN RPAREN INT
LBRACE

IF ID EQ ID
LBRACE
ID MOD ID SEMI
ID DIVIDE ID SEMI
RETURN ID SEMI
RBRACE

RBRACE
```

test_loops.tokens

```
FUNC ID LPAREN RPAREN INT LBRACE
INT ID SEMI
ID ASSIGN INTLIT SEMI
WHILE ID LT INTLIT LBRACE ID ASSIGN INTLIT SEMI RBRACE
WHILE ID LT INTLIT LBRACE ID ASSIGN INTLIT SEMI ID SEMI RBRACE
WHILE ID GT INTLIT LBRACE ID ASSIGN INTLIT SEMI ID SEMI CONTINUE SEMI
```

```
RBRACE
WHILE ID GT INTLIT LBRACE ID ASSIGN INTLIT SEMI ID SEMI BREAK SEMI
RBRACE
FOR ID ASSIGN INTLIT SEMI ID LT INTLIT SEMI ID INCR LBRACE ID SEMI
RBRACE
FOR ID ASSIGN INTLIT SEMI ID LT INTLIT SEMI ID INCR LBRACE ID SEMI
CONTINUE SEMI RBRACE
FOR ID ASSIGN INTLIT SEMI ID LT INTLIT SEMI ID INCR LBRACE ID SEMI
BREAK SEMI RBRACE
WHILE ID GT INTLIT LBRACE
WHILE ID LT INTLIT LBRACE ID SEMI RBRACE
RBRACE
WHILE ID GT INTLIT LBRACE
FOR ID ASSIGN INTLIT SEMI ID LT INTLIT SEMI ID INCR LBRACE ID SEMI
RBRACE
RBRACE
FOR ID ASSIGN INTLIT SEMI ID LT INTLIT SEMI ID INCR LBRACE
WHILE ID LT INTLIT LBRACE ID ASSIGN INTLIT SEMI RBRACE
RBRACE
FOR ID ASSIGN INTLIT SEMI ID LT INTLIT SEMI ID INCR LBRACE
FOR ID ASSIGN INTLIT SEMI ID LT INTLIT SEMI ID INCR LBRACE ID SEMI
RBRACE
RBRACE
RBRACE
```

test_map_decl_assign.tokens

```
FUNC ID LPAREN RPAREN INT LBRACE MAP LT STRING COMMA INT GT ID SEMI
ID ASSIGN LBRACKET STRLIT COLON INTLIT COMMA STRLIT COLON INTLIT
RBRACKET SEMI RBRACE
```

test_primitive_decl_assign.tokens

```
FUNC ID LPAREN RPAREN INT LBRACE STRING ID SEMI CHAR ID SEMI BOOL ID
SEMI INT ID SEMI DOUBLE ID SEMI ID ASSIGN STRLIT SEMI ID ASSIGN
CHARLIT SEMI ID ASSIGN TRUE SEMI ID ASSIGN INTLIT SEMI ID ASSIGN
DUBLIT SEMI RBRACE
```

test_struct.tokens

```
STRUCT STRUCTID LBRACE
FIELD INT COMMA
FIELD BOOL
RBRACE
STRUCT STRUCTID LBRACE
FIELD INT COMMA
FIELD STRING
RBRACE
FUNC ID LPAREN RPAREN INT LBRACE
STRUCTID ID SEMI
ID ASSIGN LBRACE FIELD INTLIT COMMA FIELD INTLIT RBRACE SEMI
RBRACE
FUNC ID LPAREN RPAREN STRUCTID LBRACE
RETURN SEMI
RBRACE
```

test_struct_decl_assign.tokens

```
FUNC ID LPAREN RPAREN INT LBRACE STRUCTID ID SEMI ID ASSIGN LBRACE
FIELD INTLIT RBRACE SEMI RBRACE
```

test_vdecl.tokens

```
INT ID SEMI
DOUBLE ID SEMI
BOOL ID SEMI
CHAR ID SEMI
STRING ID SEMI
MAP LT INT COMMA INT GT ID SEMI
GRAPH LT STRUCTID COMMA STRUCTID GT ID SEMI
STRUCTID ID SEMI
```

test_ast.sh

```
#!/bin/bash

AST_TEST_DIR='ast_tests/'
SUCCESS_COUNT=0
FAIL_COUNT=0
```

```
for x in `ls $AST_TEST_DIR | grep tokens`; do
  # run menhir
  cat $AST_TEST_DIR$x | tr '\n' ' ' | echo -e "$(cat -)" | menhir
--interpret --interpret-show-cst parser.mly > output
  # compare output to expected output
  TEST_NAME=`echo $x | cut -d'.' -f1`
  # DIFF=$(diff output $AST_TEST_DIR$TEST_NAME.out)
  if [[ `cat output | grep REJECT` != ""  && $x =~ "test" ]]; then
    echo $TEST_NAME FAIL
    FAIL_COUNT=$(($FAIL_COUNT + 1))
  elif [[ `cat output | grep REJECT` == "" && $x =~ "fail" ]]; then
    echo $TEST_NAME FAIL
    FAIL_COUNT=$(($FAIL_COUNT + 1))
  else
    SUCCESS_COUNT=$(($SUCCESS_COUNT + 1))
  fi
done
rm output
echo $SUCCESS_COUNT TESTS SUCCEEDED
echo $FAIL_COUNT TESTS FAILED
```

demos

dijkstra_length.gae

```
struct Int {
  value: int
}

func get_closest_node(map<Int, bool> visited, map<Int, int> dist) Int
{
  int lowest_so_far;
  Int closest_node;
  Int[] dist_keys;
  int i;
  /* loop vdecls */
  Int node;
  bool found_closest_node;

  found_closest_node := false;
  lowest_so_far := 1000000;
  dist_keys := getKeys(dist);

  for i := 0; i < lena(dist_keys); i = i + 1 {
    node = dist_keys[i];
    if dist[node] < lowest_so_far {
      if !visited[node] {
        found_closest_node = true;
        lowest_so_far = dist[node];
        closest_node = node;
      }
    }
  }
  if !found_closest_node {
    closest_node = {value: -1};
  }
  return closest_node;

}

/* returns total path weight of shortest path */
```

```
func get_shortest_path_length(graph<Int, Int> g, Int s, Int d) int {
  /* dijkstra's */
  Int[] nodes;
  int node_count;
  map<Int, bool> visited;
  map<Int, int> dist;
  int i;
  /* for-loop vdecls */
  int j;
  Int node;
  Int node2;
  edge<Int, Int>[] edges;
  Int dst;
  Int val;
  edge<Int, Int> e;
  int cur_best;
  int next_best;

  nodes := getNodes(g);
  node_count := lena(nodes);
  visited := map_init();
  dist := map_init();

  for i := 0; i < node_count; i++ {
    node = nodes[i];
    dist[nodes[i]] = 10000;
    visited[nodes[i]] = false;
  }
  dist[s] = 0;

  /* instead of using a priority queue, we get the closest node on
each loop */
  for i := 0; i < node_count; i++ {
    node = get_closest_node(visited, dist);
    if node.value != -1 {
      visited[node] = true;
      edges = getEdges(g);
      for j := 0; j < lena(edges); j++ {
```

```
        e = edges[j];
        node2 = getSrc(e);
        if node == node2 {
          dst = getDst(e);
          val = getVal(e);
          cur_best = dist[dst];
          next_best = dist[node] + val.value;
          if next_best < cur_best {
            dist[dst] = next_best;
          }
        }
      }
    }
  }
  return dist[d];
}

func main() int {
  graph<Int, Int> g;
  g := {
    ({value: 1},{value: 2},{value: 1}), /* syntax: (src, dest, edge)
*/
    ({value: 1},{value: 3},{value: 2}),
    ({value: 1},{value: 4},{value: 3}),
    ({value: 2},{value: 3},{value: 4}),
    ({value: 2},{value: 4},{value: 5}),
    ({value: 3},{value: 4},{value: 6}),
    ({value: 2},{value: 1},{value: 7}),
    ({value: 4},{value: 1},{value: 9}),
    ({value: 3},{value: 2},{value: 10}),
    ({value: 4},{value: 2},{value: 11}),
    ({value: 4},{value: 3},{value: 12})
  };
  printi(get_shortest_path_length(g, {value: 3}, {value: 1})); /*
should return 15 */
  g = {
    ({value: 1},{value: 2},{value: 1}),
    ({value: 1},{value: 3},{value: 10}),
```

```
    ({value: 2},{value: 3},{value: 30}),
    ({value: 2},{value: 4},{value: 5}),
    ({value: 3},{value: 4},{value: 1})
  };
  printi(get_shortest_path_length(g, {value: 1}, {value: 4})); /*
should return 6 */
  g = {
    ({value: 1}, {value: 2}, {value: 2}),
    ({value: 1}, {value: 3}, {value: 4}),
    ({value: 2}, {value: 3}, {value: 1}),
    ({value: 2}, {value: 4}, {value: 4}),
    ({value: 2}, {value: 5}, {value: 2}),
    ({value: 3}, {value: 5}, {value: 3}),
    ({value: 5}, {value: 4}, {value: 3}),
    ({value: 4}, {value: 6}, {value: 2}),
    ({value: 5}, {value: 6}, {value: 2})
  };
  printi(get_shortest_path_length(g, {value: 1}, {value: 6})); /*
should return 6 */
  return 0;
}
```

dijkstra_path.gae

```
struct Int {
  value: int
}

func get_closest_node(map<Int, bool> visited, map<Int, int> dist) Int
{
  int lowest_so_far;
  Int closest_node;
  Int[] dist_keys;
  int i;
  /* loop vdecls */
  Int node;
  bool found_closest_node;

  found_closest_node := false;
  lowest_so_far := 1000000000;
```

```
  dist_keys := getKeys(dist);

  for i := 0; i < lena(dist_keys); i = i + 1 {
    node = dist_keys[i];
    if dist[node] < lowest_so_far {
      if !visited[node] {
        found_closest_node = true;
        lowest_so_far = dist[node];
        closest_node = node;
      }
    }
  }
  if !found_closest_node {
    closest_node = {value: -1};
  }
  return closest_node;

}

/* returns total path weight of shortest path */
func get_shortest_path(graph<Int, Int> g, Int s, Int d) Int[] {
  /* dijkstra's */
  Int[] nodes;
  int node_count;
  map<Int, bool> visited;
  map<Int, int> dist;
  map<Int, Int> prev;
  int i;
  /* for-loop vdecls */
  int j;
  Int node;
  Int node2;
  edge<Int, Int>[] edges;
  Int dst;
  Int val;
  edge<Int, Int> e;
  int cur_best;
  int next_best;
```

```
  Int[] path;

  nodes := getNodes(g);
  node_count := lena(nodes);
  visited := map_init();
  dist := map_init();
  prev := map_init();

  for i := 0; i < node_count; i++ {
    node = nodes[i];
    dist[nodes[i]] = 1000000000;
    visited[nodes[i]] = false;
  }
  dist[s] = 0;
  prev[s] = {value: -1};

  /* instead of using a priority queue, we get the closest node on
each loop */
  for i := 0; i < node_count; i++ {
    node = get_closest_node(visited, dist);
    if node.value != -1 {
      visited[node] = true;
      edges = getEdges(g);
      for j := 0; j < lena(edges); j++ {
        e = edges[j];
        node2 = getSrc(e);
        if node == node2 {
          dst = getDst(e);
          val = getVal(e);
          cur_best = dist[dst];
          next_best = dist[node] + val.value;
          if next_best < cur_best {
            dist[dst] = next_best;
            prev[dst] = node;
          }
        }
      }
    }
```

```
  }

  /* run through prev to get the path */
  path := arr_init();
  append(path, d);
  node = prev[d];
  while node.value != -1 {
    append(path, node);
    node = prev[node];
  }

  return path;
}

func print_path(Int[] path) int {
  Int node;
  int i;
  for i := lena(path)-1; i >= 0; i-- {
    node = path[i];
    printi(node.value);
  }
  return 0;
}

func main() int {
  graph<Int, Int> g;
  Int[] path;
  Int node;
  int i;

  g := {
    ({value: 1},{value: 2},{value: 1}), /* syntax: (src, dest, edge)
*/
    ({value: 1},{value: 3},{value: 2}),
    ({value: 1},{value: 4},{value: 3}),
    ({value: 2},{value: 3},{value: 4}),
    ({value: 2},{value: 4},{value: 5}),
    ({value: 3},{value: 4},{value: 6}),
```

```
  ({value: 2},{value: 1},{value: 7}),
  ({value: 4},{value: 1},{value: 9}),
  ({value: 3},{value: 2},{value: 10}),
  ({value: 4},{value: 2},{value: 11}),
  ({value: 4},{value: 3},{value: 12})
};
path := get_shortest_path(g, {value: 3}, {value: 1});
prints("Path 1");
print_path(path);

g = {
  ({value: 1},{value: 2},{value: 1}),
  ({value: 1},{value: 3},{value: 10}),
  ({value: 2},{value: 3},{value: 30}),
  ({value: 2},{value: 4},{value: 5}),
  ({value: 3},{value: 4},{value: 1})
};
path = get_shortest_path(g, {value: 1}, {value: 4});
prints("Path 2");
print_path(path);

g = {
  ({value: 1}, {value: 2}, {value: 2}),
  ({value: 1}, {value: 3}, {value: 4}),
  ({value: 2}, {value: 3}, {value: 1}),
  ({value: 2}, {value: 4}, {value: 4}),
  ({value: 2}, {value: 5}, {value: 2}),
  ({value: 3}, {value: 5}, {value: 3}),
  ({value: 5}, {value: 4}, {value: 3}),
  ({value: 4}, {value: 6}, {value: 2}),
  ({value: 5}, {value: 6}, {value: 2})
};
path = get_shortest_path(g, {value: 1}, {value: 6});
prints("Path 3");
print_path(path);

g := graph_init();
for i := 1; i <= 50; i++ {
```

```
    addEdge(g, ({value: i}, {value: 3*i+1}, {value: 1}));
    addEdge(g, ({value: 2*i}, {value: i}, {value: 1}));
  }
  path = get_shortest_path(g, {value: 50}, {value: 1});
  prints("Path 4 (Collatz conjecture simulation)");
  print_path(path);

  return 0;
}
```

helloworld.gae

```
func main() int {
     prints("Hello World!");
}
```

median_of_two_arrays.gae

```
func main() int {
  int[] var;
  int[] var1;
  int[] merged;
  int temp;
  double median;
  int i;
  int j;
  bool a;
  bool b;

  var := [1,3,4,5,6,7];
  var1 := [3,4,7,9,10];

  i := 0;
  j := 0;
  merged := arr_init();

  while((i < lena(var)) && (j < lena(var1))){
    if var[i] <= var1[j] {
      append(merged, var[i]);
      i++;
    }
```

```
    else{
      append(merged, var1[j]);
      j++;
    }
  }

  if i < lena(var) {
    for i; i < lena(var); i++ {
      append(merged, var[i]);
    }
  }
  else{
    for j; j < lena(var1); j++ {
      append(merged, var1[j]);
    }
  }

  if (lena(merged) % 2) == 0 {
    temp = lena(merged) / 2;
    temp = (merged[temp] + merged[temp - 1]) / 2;
    median = double_of_int(temp);
  }
  else {
    temp = merged[lena(merged) / 2];
    median = double_of_int(temp);
  }

  printd(median);
  return 0;
}
```

Main Code

scanner.mll

```
(* Ocamllex scanner for GaE *)

{ open Parser }

let Exp = ['e' 'E']['+' '-']?['0'-'9']+

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"     { comment lexbuf }           (* Comments *)
| '('      { LPAREN }
| ')'      { RPAREN }
| '['      { LBRACKET }
| ']'      { RBRACKET }
| '{'      { LBRACE }
| '}'      { RBRACE }
| ':'      { COLON }
| ';'      { SEMI }
| ','      { COMMA }
| '.'      { PERIOD }
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '%'      { MOD }
| "++"     { INCR }
| "--"     { DECR }
| "+."     { DPLUS }
| "-."     { DMINUS }
| "*."     { DTIMES }
| "/."     { DDIVIDE }
| "%."     { DMOD }
| '='      { REASSIGN }
| ":="     { ASSIGN }
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
```

```
| "<="     { LEQ }
| ">"      { GT }
| ">="     { GEQ }
| "&&"     { AND }
| "||"     { OR }
| "!"      { NOT }
| "if"     { IF }
| "elseif" { ELSEIF }
| "else"   { ELSE }
| "for"    { FOR }
| "while"  { WHILE }
| "return" { RETURN }
| "continue" { CONTINUE }
| "in"     { IN }
| "func"   { FUNC }
| "struct" { STRUCT }
| "int"    { INT }
| "double" { DOUBLE }
| "bool"   { BOOL }
| "char"   { CHAR }
| "string" { STRING }
| "graph"  { GRAPH }
| "edge"   { EDGE }
| "map"    { MAP }
| "break"  { BREAK }
| "true"   { TRUE }
| "false"  { FALSE }
| ['0'-'9']+ as lxm { INTLIT(int_of_string lxm) }
| '\'' [ ^'\'' ]  '\'' as charlit  { CHARLIT(String.sub charlit 1
((String.length charlit) - 2)) }
| '"' [ ^'"'  ]* '"'   as strlit { STRLIT(String.sub strlit 1
((String.length strlit) - 2)) }
| ( '.' ['0'-'9']+ Exp? | ['0'-'9']+ ( '.' ['0'-'9']* Exp? | Exp)) as
lxm { DUBLIT(float_of_string lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*[':'] as lxm {
FIELD(String.sub lxm 0 ((String.length lxm) - 1)) }
| ['a'-'z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| ['A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { STRUCTID(lxm) }
```

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

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

parser.mly

```
/* Ocamlyacc parser for MicroC */

%{
open Ast
%}

%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA PERIOD
%token PLUS MINUS TIMES DIVIDE ASSIGN NOT
%token EQ NEQ LT LEQ GT GEQ AND OR
%token RETURN IF ELSEIF ELSE FOR WHILE
%token INT BOOL DOUBLE CHAR STRING STRUCT MAP GRAPH EDGE
%token MOD DPLUS DMINUS DTIMES DDIVIDE DMOD IN
%token REASSIGN FUNC COLON INCR DECR CONTINUE
%token BREAK LBRACKET RBRACKET TRUE FALSE
%token <int> INTLIT
%token <float> DUBLIT
%token <string> CHARLIT
%token <string> STRLIT
%token <string> ID
%token <string> STRUCTID
%token <string> FIELD
%token EOF

%right ASSIGN REASSIGN
%left IN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
```

```
%left PLUS MINUS DPLUS DMINUS
%left TIMES DIVIDE MOD DTIMES DDIVIDE DMOD
%right NOT NEG

%start program
%type <Ast.program> program

%%

program:
  decls EOF { $1 }

decls:
   /* nothing */ { [], [], [] }
 | decls vdecl { ($2 :: fst_of_3 $1), snd_of_3 $1, thd_of_3 $1 }
 | decls sdecl { fst_of_3 $1, ($2 :: snd_of_3 $1), thd_of_3 $1 }
 | decls fdecl { fst_of_3 $1, snd_of_3 $1, ($2 :: thd_of_3 $1) }

fdecl:
  FUNC ID LPAREN formals_opt RPAREN typ LBRACE vdecl_list stmt_list
RBRACE
     { { typ = $6;
       fname = $2;
       formals = $4;
       locals = List.rev $8;
       body = List.rev $9 } }

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

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

vdecl_list:
    /* nothing */     { [] }
  | vdecl_list vdecl  { $2 :: $1 }
```

```
typ:
    base_typ_or_struct { $1 }
  | MAP LT base_typ_or_struct COMMA base_typ_or_struct GT { Map($3,
$5) } /* map<int, int> */
  | GRAPH LT STRUCTID COMMA STRUCTID GT { Graph(StructType($3),
StructType($5)) } /* graph<Int, Int> */
  | base_typ_or_struct LBRACKET RBRACKET { Array($1) } /* int[] */

base_typ: /* non-container types only */
    INT { Int }
  | DOUBLE { Double }
  | BOOL { Bool }
  | CHAR { Char }
  | STRING { String }

base_typ_or_struct:
    base_typ { $1 }
  | STRUCTID { StructType($1) }
  | EDGE LT STRUCTID COMMA STRUCTID GT { Edge(StructType($3),
StructType($5)) }

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

sdecl:
    STRUCT STRUCTID LBRACE struct_field_list RBRACE { ($2, $4) }
/*(struct_name, list of field:type in the struct)*/

struct_field_list:
    struct_field                          { [$1] }
  | struct_field_list COMMA struct_field    { $3 :: $1 }

struct_field:
    FIELD base_typ { ($1, $2) } /* returns (field_name, field_type)
*/

stmt_list:
```

```
    stmt            { [$1] }
  | stmt_list stmt  { $2 :: $1 }

loop_stmt_list:
    stmt                                { [$1] }
  | loop_stmt_list stmt                 { $2 :: $1 }
  | loop_stmt_list continue_or_break    { $2 :: $1 }

continue_or_break:
    CONTINUE SEMI   { Continue }
  | BREAK SEMI      { Break }

stmt:
    expr SEMI { Expr $1 }
  | RETURN SEMI { Return Noexpr }
  | RETURN expr SEMI { Return $2 }
  | LBRACE stmt_list RBRACE { Block(List.rev $2) }
  | IF expr LBRACE stmt_list RBRACE { If($2, $4, [Block([])]) }
  | IF expr LBRACE stmt_list RBRACE ELSE LBRACE stmt_list RBRACE {
If($2, $4, $8) }
  | IF expr LBRACE stmt_list RBRACE elseif_list { IfElseifs($2, $4,
$6, [Block([])]) }
  | IF expr LBRACE stmt_list RBRACE elseif_list ELSE LBRACE stmt_list
RBRACE { IfElseifs($2, $4, $6, $9)}
  | FOR expr SEMI expr SEMI expr LBRACE loop_stmt_list RBRACE
     { For($2, $4, $6, $8) }
  | WHILE expr LBRACE loop_stmt_list RBRACE { While($2, $4) }

elseif:
  ELSEIF expr LBRACE stmt_list RBRACE { ($2, $4) }

elseif_list:
    elseif { [$1] }
  | elseif_list elseif { $2 :: $1 }

expr:
    INTLIT              { IntLit($1) }
  | DUBLIT              { DubLit($1) }
```

```
  | CHARLIT             { CharLit($1) }
  | STRLIT              { StrLit($1) }
  | TRUE                { True }
  | FALSE               { False }
  | ID                  { Id($1) }
  | expr PLUS     expr  { Binop($1, Add,   $3) }
  | expr DPLUS    expr  { Binop($1, DAdd,  $3) }
  | expr MINUS    expr  { Binop($1, Sub,   $3) }
  | expr DMINUS   expr  { Binop($1, DSub,  $3) }
  | expr TIMES    expr  { Binop($1, Mult,  $3) }
  | expr DTIMES   expr  { Binop($1, DMult, $3) }
  | expr DIVIDE   expr  { Binop($1, Div,   $3) }
  | expr DDIVIDE  expr  { Binop($1, DDiv,  $3) }
  | expr MOD      expr  { Binop($1, Mod,   $3) }
  | expr DMOD     expr  { Binop($1, DMod,  $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) }
  | expr IN       expr  { Binop($1, In,    $3) }
  | MINUS expr %prec NEG { Unop(Neg, $2) }
  | NOT expr          { Unop(Not, $2) }
  | ID INCR           { Assign(Id($1), (Binop(Id($1), Add,
IntLit(1)))) }
  | ID DECR           { Assign(Id($1), (Binop(Id($1), Sub,
IntLit(1)))) }
  | assignable ASSIGN expr   { Assign($1, $3) }
  | assignable REASSIGN expr   { Reassign($1, $3) }
  | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
  | LPAREN expr RPAREN { $2 }
  /* Struct literal */
  | struct_lit { $1 }
  /* Map literal */
  | LBRACKET kv_list RBRACKET { MapLit($2) }
```

```
  /* Graph literal */
  | LBRACE edge_list RBRACE { GraphLit($2) }
  /* Edge literal */
  | edge_lit { $1 }
  /* Array literal */
  | LBRACKET actuals_list RBRACKET { ArrayLit($2) }
  /* Access array/map */
  | ID LBRACKET expr RBRACKET      { Index($1,$3) }
  /* Access struct field */
  | ID PERIOD ID          { StructFieldAccess($1, $3) }

assignable:
    ID                             { Id($1) }
  | ID LBRACKET expr RBRACKET      { Index($1,$3) }
  | ID PERIOD ID                   { StructFieldAccess($1, $3) }

struct_lit:
  LBRACE field_list RBRACE { StructLit($2) }

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

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

field_list:
    field                     { [$1] }
  | field_list COMMA field    { $3 :: $1 }

field:
  FIELD expr             { ($1, $2) }

kv_list:
    kv                  { [$1] }
  | kv_list COMMA kv    { $3 :: $1 }
```

```
kv:
  key_lit COLON expr      { ($1, $3) }

key_lit:
    STRLIT { StrLit($1) }
  | INTLIT { IntLit($1) }
  | CHARLIT { CharLit($1) }
  | struct_lit { $1 }
  | ID      { Id($1) }

/* TODO: (note) disallow non-fully connected graphs */
edge_list:
    edge_lit     { [$1] }
  | edge_list COMMA edge_lit { $3 :: $1 }

edge_lit:
  LPAREN node_lit COMMA node_lit COMMA node_lit RPAREN { EdgeLit($2,
$4, $6) }

/* We can probably just replace node_lit with expr to allow any type
to be node/edge value? */
node_lit:
    struct_lit  { $1 }
  | ID          { Id($1) }
```

ast.ml

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

type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater
| Geq |
          And | Or | Mod | DAdd | DSub | DMult | DDiv | DMod | In

(* Negation equals = Notequals | Negation op = not op *)
type uop = Neg | Not | Incr | Decr

(* define the range of types available *)
type typ =
    Int
```

```
  | Double
  | Bool
  | Char
  | String
  | StructType of string
  | Map of typ * typ
  | Graph of typ * typ
  | Edge of typ * typ
  | Array of typ

type bind = typ * string

type struct_field = string * typ
type struct_decl = string * struct_field list

type expr =
    IntLit of int
  | DubLit of float
  | True
  | False
  | CharLit of string
  | StrLit of string
  | StructLit of (string * expr) list
  | MapLit of (expr * expr) list
  | GraphLit of expr list
  | EdgeLit of expr * expr * expr
  | ArrayLit of expr list
  | Id of string
  | Index of string * expr
  | StructFieldAccess of string * string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Assign of expr * expr
  | Reassign of expr * expr
  | Call of string * expr list
  | Noexpr

type stmt =
```

```
    Block of stmt list
  | Expr of expr
  | Return of expr
  | If of expr * stmt list * stmt list
  | IfElseifs of expr * stmt list * (expr * stmt list) list * stmt
list
  | For of expr * expr * expr * stmt list
  | While of expr * stmt list
  | Continue
  | Break

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

(* vdecl_list, sdecl_list, fdecl_list *)
type program = bind list * struct_decl list * func_decl list

(* Pretty-printing functions *)

let string_of_op = function
    Add -> "+"
  | DAdd -> "+."
  | Sub -> "-"
  | DSub -> "-."
  | Mult -> "*"
  | DMult -> "*."
  | Div -> "/"
  | DDiv -> "/."
  | Mod -> "%"
  | DMod -> "%."
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
```

```
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "&&"
  | Or -> "||"
  | In -> "in"


let string_of_uop = function
    Neg -> "-"
  | Not -> "!"
  | Incr -> "++"
  | Decr -> "--"

let rec string_of_expr = function
    IntLit(l) -> string_of_int l
  | DubLit(d) -> string_of_float d
  | True -> "true"
  | False -> "false"
  | CharLit(c) -> c
  | StrLit(s) -> s
  | ArrayLit(l) -> "[" ^ String.concat "," (List.map string_of_expr
(List.rev l)) ^ "]"
  | StructLit(l) -> "{" ^ String.concat "," (List.map (fun (f, v) ->
f ^ ":" ^ (string_of_expr v)) (List.rev l)) ^ "}"
  | MapLit(l) -> "[" ^ String.concat "," (List.map (fun (k, v) ->
(string_of_expr k) ^ ":" ^ (string_of_expr v)) (List.rev l)) ^ "]"
  | GraphLit(l) -> "{" ^ String.concat ", " (List.map string_of_expr
(List.rev l)) ^ "}"
  | EdgeLit(a, b, c) -> "(" ^ (string_of_expr a) ^ ", " ^
(string_of_expr b) ^ ", " ^ (string_of_expr c) ^ ")"
  | Id(s) -> s
  | Index(s, e) -> s ^ "[" ^ (string_of_expr e) ^ "]"
  | StructFieldAccess (s, t) -> s ^ "." ^ t
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr
e2
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
```

```
  | Assign(l, r) -> string_of_expr l ^ " := " ^ string_of_expr r
  | Reassign(l, r) -> string_of_expr l ^ " = " ^ string_of_expr r
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | Noexpr -> ""

let rec string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt (List.rev
stmts)) ^ "}\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
  | If(e, s, [Block([])]) -> "if (" ^ string_of_expr e ^ ")\n  " ^
      String.concat "  " (List.map string_of_stmt (List.rev s))
  | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n  " ^
      String.concat "  " (List.map string_of_stmt (List.rev s1)) ^
      "else\n  " ^ String.concat "  " (List.map string_of_stmt
(List.rev s2))
  | IfElseifs(e1, s1, l, [Block([])]) -> "if (" ^ string_of_expr e1 ^
")\n  " ^
      String.concat "  " (List.map string_of_stmt (List.rev s1)) ^
String.concat ""
      (List.map (fun (e2, s2) -> ("elseif (" ^ string_of_expr e2 ^
")\n  " ^ String.concat "  " (List.map string_of_stmt (List.rev
s2)))) l)
  | IfElseifs(e1, s1, l, s3) ->  "if (" ^ string_of_expr e1 ^ ")\n  "
^
      String.concat "  " (List.map string_of_stmt (List.rev s1)) ^
String.concat ""
      (List.map (fun (e2, s2) -> ("elseif (" ^ string_of_expr e2 ^
")\n  " ^ String.concat "  " (List.map string_of_stmt (List.rev
s2)))) l) ^
      "else\n  " ^ String.concat "  " (List.map string_of_stmt
(List.rev s3))
  | For(e1, e2, e3, s) ->
      "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ;
" ^
      string_of_expr e3  ^ ") " ^ String.concat "" (List.map
```

```
string_of_stmt (List.rev s))
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^
String.concat "" (List.map string_of_stmt (List.rev s))
  | Continue -> "continue;"
  | Break -> "break;"

let rec string_of_typ = function
    Int -> "int"
  | Bool -> "bool"
  | String -> "str"
  | Double -> "double"
  | Char -> "char"
  | Array(t) -> (string_of_typ t) ^ "[]"
  | StructType(t) -> t
  | Map(a, b) -> "map<" ^ (string_of_typ a) ^ ", " ^ (string_of_typ
b) ^ ">"
  | Graph(a, b) -> "graph<" ^ (string_of_typ a) ^ ", " ^
(string_of_typ b) ^ ">"
  | Edge(a, b) -> "edge<" ^ (string_of_typ a) ^ ", " ^ (string_of_typ
b) ^ ">"

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

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

let string_of_sdecl (struct_name, fields) =
  struct_name ^ "{" ^ String.concat "," (List.map (fun (f, t) -> f ^
      ":" ^ (string_of_typ t)) fields)  ^ "}"

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

```
  String.concat "\n" (List.map string_of_sdecl structs) ^ "\n" ^
  String.concat "\n" (List.map string_of_fdecl funcs)

(* Helper functions *)
let fst_of_3 (a,_,_) = a
let snd_of_3 (_,a,_) = a
let thd_of_3 (_,_,a) = a
```

sast.ml

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

open Ast

type sexpr = typ * sx
and sx =
    SIntLit of int
  | SDubLit of float
  | SCharLit of string
  | SStrLit of string
  | SStructLit of (string * sexpr) list
  | SMapLit of (sexpr * sexpr) list
  | SGraphLit of sexpr list
  | SEdgeLit of sexpr * sexpr * sexpr
  | SArrayLit of sexpr list
  | SId of string
  | SArrayIndex of string * sexpr
  | SMapIndex of string * sexpr
  | SStructFieldAccess of string * string
  | SBinop of sexpr * op * sexpr
  | SUnop of uop * sexpr
  | SAssign of sexpr * sexpr
  | SReassign of sexpr * sexpr
  | SCall of string * sexpr list
  | STrue
  | SFalse
  | SNoexpr
```

```
type sstmt =
    SBlock of sstmt list
  | SExpr of sexpr
  | SReturn of sexpr
  | SIf of sexpr * sstmt list * sstmt list
  | SIfElseifs of sexpr * sstmt list * (sexpr * sstmt list) list *
sstmt list
  | SFor of sexpr * sexpr * sexpr * sstmt list
  | SWhile of sexpr * sstmt list
  | SContinue
  | SBreak

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

type sprogram = bind list * struct_decl list * sfunc_decl list

(* Pretty-printing functions *)

let rec string_of_sexpr (t, e) =
  "(" ^ string_of_typ t ^ " : " ^ (match e with
    SIntLit(l) -> string_of_int l
  | SDubLit(l) -> string_of_float l
  | STrue -> "true"
  | SFalse -> "false"
  | SCharLit(c) -> c
  | SStrLit(s) -> s
  | SArrayLit(l) -> "[" ^ String.concat "," (List.map string_of_sexpr
(List.rev l)) ^ "]"
  | SStructLit(l) -> "{" ^ String.concat "," (List.map (fun (f, v) ->
f ^ ":" ^ (string_of_sexpr v)) (List.rev l)) ^ "}"
  | SMapLit(l) -> "[" ^ String.concat "," (List.map (fun (k, v) ->
(string_of_sexpr k) ^ ":" ^ (string_of_sexpr v)) (List.rev l)) ^ "]"
```

```
  | SGraphLit(l) -> "{" ^ String.concat ", " (List.map
string_of_sexpr (List.rev l)) ^ "}"
  | SEdgeLit(a, b, c) -> "(" ^ (string_of_sexpr a) ^ ", " ^
(string_of_sexpr b) ^ ", " ^ (string_of_sexpr c) ^ ")"
  | SId(s) -> s
  | SArrayIndex(s, e) | SMapIndex(s, e) -> s ^ "[" ^ (string_of_sexpr
e) ^ "]"
  | SStructFieldAccess(s, t) -> s ^ "." ^ t
  | SBinop(e1, o, e2) ->
      string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^
string_of_sexpr e2
  | SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
  | SAssign(l, r) -> string_of_sexpr l ^ " := " ^ string_of_sexpr r
  | SReassign(l, r) -> string_of_sexpr l ^ " = " ^ string_of_sexpr r
  | SCall(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^
")"
  | SNoexpr -> ""
                        ) ^ ")"

let rec string_of_sstmt = function
    SBlock(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_sstmt (List.rev
stmts)) ^ "}\n"
  | SExpr(expr) -> string_of_sexpr expr ^ ";\n";
  | SReturn(expr) -> "return " ^ string_of_sexpr expr ^ ";\n";
  | SIf(e, s, [SBlock([])]) -> "if (" ^ string_of_sexpr e ^ ")\n  " ^
      String.concat "  " (List.map string_of_sstmt (List.rev s))
  | SIf(e, s1, s2) ->  "if (" ^ string_of_sexpr e ^ ")\n  " ^
      String.concat "  " (List.map string_of_sstmt (List.rev s1)) ^
      "else\n  " ^ String.concat "  " (List.map string_of_sstmt
(List.rev s2))
  | SIfElseifs(e1, s1, l, [SBlock([])]) -> "if (" ^ string_of_sexpr
e1 ^ ")\n  " ^
      String.concat "  " (List.map string_of_sstmt (List.rev s1)) ^
String.concat ""
      (List.map (fun (e2, s2) -> ("elseif (" ^ string_of_sexpr e2 ^
")\n  " ^ String.concat "  " (List.map string_of_sstmt (List.rev
```

```
s2)))) l)
  | SIfElseifs(e1, s1, l, s3) -> "if (" ^ string_of_sexpr e1 ^ ")\n
" ^
      String.concat "  " (List.map string_of_sstmt (List.rev s1)) ^
String.concat ""
      (List.map (fun (e2, s2) -> ("elseif (" ^ string_of_sexpr e2 ^
")\n  " ^ String.concat "  " (List.map string_of_sstmt (List.rev
s2)))) l) ^
      "else\n  " ^ String.concat "  " (List.map string_of_sstmt
(List.rev s3))
  | SFor(e1, e2, e3, s) ->
      "for (" ^ string_of_sexpr e1  ^ " ; " ^ string_of_sexpr e2 ^ "
; " ^
      string_of_sexpr e3  ^ ") " ^ String.concat "" (List.map
string_of_sstmt (List.rev s))
  | SWhile(e, s) -> "while (" ^ string_of_sexpr e ^ ") " ^
String.concat "" (List.map string_of_sstmt (List.rev s))
  | SContinue -> "continue;"
  | SBreak -> "break;"

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

let string_of_ssdecl (struct_name, fields) =
  struct_name ^ "{" ^ String.concat "," (List.map (fun (f, t) -> f ^
      ":" ^ (string_of_typ t)) fields)  ^ "}"

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

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

```
  String.concat "\n" (List.map string_of_sfdecl funcs)
```

semant.ml

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

open Ast
open Sast

module StringMap = Map.Make(String)

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

   Check each global variable, then check each function *)

let check (globals, structs, functions) =

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

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

  check_binds "global" globals;

  (**** Check struct declarations ****)
  (* structs are only ever declared in the global scope *)
  let check_struct (structs : struct_decl list) =

    let rec dups = function
        [] -> ()
```

```
      |    ((n1,_) :: (n2,_) :: _) when n1 = n2 ->
       raise (Failure ("duplicate struct " ^ n1))
      | _ :: t -> dups t
    in
    (* Verify no duplicate struct names *)
    dups (List.sort (fun (a,_) (b,_) -> compare a b) structs);
    (* Verify no duplicate struct field names *)
    List.iter (fun (_, field_list) -> dups (List.sort (fun (a,_)
(b,_) -> compare a b) field_list)) structs;
  in

  check_struct structs;

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

  (* Create struct symbol table *)
  let struct_decls =
    let add_struct map (name, fields) = StringMap.add name fields map
    in List.fold_left add_struct StringMap.empty structs
  in

  let find_struct s =
    try StringMap.find s struct_decls
    with Not_found -> raise (Failure ("unrecognized struct " ^ s))
  in

  (* Construct string by joining "field:type_of_expr" of fields in
sorted order with ',' *)
  let string_of_fields (field_list : struct_field list) =
    String.concat "," (List.map (fun (field, typ) -> field ^
(string_of_typ typ)) (List.sort (fun (a,_) (b,_) -> compare a b)
field_list))
  in
  (* Maps string_of_fields -> struct_name *)
  let struct_decls_rev =
    let add_struct map (name, fields) = StringMap.add
(string_of_fields fields) name map
    in List.fold_left add_struct StringMap.empty structs
```

```
  in

  (* Check if fields_str exists in struct table, if it does return
the struct name, else return fields_str *)
  let get_struct_id fields_str =
    try StringMap.find fields_str struct_decls_rev
    with Not_found -> fields_str
  in

  (* Initialize function symbol table with built-in functions: no
bodies *)
  let built_in_fn_decls =
    let add_bind map (name, args, ret_type) = StringMap.add name {
      typ = ret_type;
      fname = name;
      formals = args;
      locals = []; body = [] } map
    in List.fold_left add_bind StringMap.empty [
      ("printi", [(Int, "arg")], Int);
      ("printb", [(Bool, "arg")], Int);
      ("prints", [(String, "arg")], Int);
      ("printc", [(Char, "arg")], Int);
      ("printd", [(Double, "arg")], Int);
      ("printbig", [(Int, "arg")], Int);
      (* following args have non-native types, refer to Call rule *)
      ("lena", [(Int, "arg")], Int);                    (* arg type:
array *)
      ("lenm", [(Int, "arg")], Int);                    (* arg type:
map *)
      ("lens", [(String, "arg")], Int);                 (* arg type:
string *)
      ("getKeys", [(Int, "map")], Int);                 (* arg type:
map, ret type: array *)
      ("map_init", [], Int);
      ("arr_init", [], Int);
      ("graph_init", [], Int);
      ("getSrc", [(Int, "edge")], Int);                 (* arg type:
edge; ret type: struct *)
```

```
      ("getDst", [(Int, "edge")], Int);                (* arg type:
edge; ret type: struct *)
      ("getVal", [(Int, "edge")], Int);                (* arg type:
edge; ret type: struct *)
      ("setSrc", [(Int, "edge"); (Int, "src")], Int);   (* arg types:
edge, src; ret type: struct *)
      ("setDst", [(Int, "edge"); (Int, "dst")], Int);   (* arg type:
edge, dst; ret type: struct *)
      ("setVal", [(Int, "edge"); (Int, "val")], Int);   (* arg type:
edge, val; ret type: struct *)
      ("append", [(Int, "arr"); (Int, "elmt")], Int);
      ("addEdge", [(Int, "graph"); (Int, "edge")], Int);(* arg type:
graph, edge; ret type: edge *)
      ("getEdges", [(Int, "graph")], Int);             (* arg type:
graph, ret type: array of edges *)
      ("getNodes", [(Int, "graph")], Int);             (* arg type:
graph, ret type: array of nodes *)
      ("int_of_double", [(Double, "arg")], Int);
      ("double_of_int", [(Int, "arg")], Double);
  ]
  in

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

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

```
functions
  in

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

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

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

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

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

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

    (* Helper functions *)
```

```
    let rec check_all_same_types l err_type = match l with
        [] -> ()
        |  (t1 :: t2 :: _) when t1 <> t2 ->
                raise (Failure ("non-duplicate " ^ err_type ^ ": t1 =
" ^ (string_of_typ t1) ^ ", t2 = " ^ (string_of_typ t2)))
        | _ :: t -> check_all_same_types t err_type
    in
    let rec check_all_diff_vals l err_type = match l with
        [] -> ()
        |  (v1 :: v2 :: _) when v1 = v2 ->
                raise (Failure ("duplicate " ^ err_type ^ ": v1 = v2
= " ^ (string_of_expr v1)))
        | _ :: t -> check_all_diff_vals t err_type
    in

    (* Return a semantically-checked expression, i.e., with a type *)
    let rec expr = function
        IntLit  l -> (Int, SIntLit l)
      | DubLit l -> (Double, SDubLit l)
      | CharLit l -> (Char, SCharLit l) (*TODO: may have error
here?*)
      | StrLit l -> (String, SStrLit l)
      | StructLit l ->
        (* Verify no duplicate struct field names *)
        check_all_diff_vals (List.sort (fun (a) (b) -> compare a b)
(List.map (fun (field, _) -> StrLit(field)) structs)) "struct
fields";
        (* Get a struct_field list by using expr *)
        let get_struct_field_list (struct_lit : (string * expr) list)
=
          List.map (fun (a,b) -> (a, fst (expr b))) struct_lit
        in
        (* construct string_of_fields *)
        let get_struct_type struct_lit =
          get_struct_id (string_of_fields (get_struct_field_list
struct_lit))
        in
        (* construct semantically checked lit *)
```

```
        let get_sstruct_lit struct_lit =
          List.map (fun (a,b) -> (a, expr b)) struct_lit
        in
        let sstruct_lit = get_sstruct_lit l
        in
        (* Verify that all expr are non-container types *)
        let check_nc_types = function
            Int -> ()
          | Bool -> ()
          | Double -> ()
          | Char -> ()
          | String -> ()
          | StructType(_) -> ()
          | _ as t -> raise (Failure ("container type found inside
struct literal " ^ string_of_typ t))
        in
        List.iter check_nc_types (List.map snd (get_struct_field_list
l));
        (* Return (StructType(StructId), SStructLit l)*)
        (StructType(get_struct_type l), SStructLit sstruct_lit)
      | MapLit l ->
        let get_key_types (map_lit : (expr * expr) list) =
          List.map (fun (k, _) -> fst (expr k)) map_lit
        in
        let get_key_values (map_lit : (expr * expr) list) =
          List.map (fun (k, _) -> k) map_lit
        in
        let get_value_types (map_lit : (expr * expr) list) =
          List.map (fun (_, v) -> fst (expr v)) map_lit
        in
        (* Verify all keys are same type *)
        check_all_same_types (get_key_types l) "key type";
        (* Verify all values are same type *)
        check_all_same_types (get_value_types l) "value type";
        (* Verify no duplicate keys *)
        check_all_diff_vals (get_key_values l) "keys";
        (* Return (Map(t1, t2), SMapLit(sl)) *)
        let t1 = List.nth (get_key_types l) 0 in
```

```
        let t2 = List.nth (get_value_types l) 0 in
        let sl = List.map (fun (k, v) -> (expr k, expr v)) l in
        (Map(t1, t2), SMapLit sl)
      | GraphLit l ->
        let get_node_types (graph_lit : expr list) =
          List.map (fun (e) -> match fst (expr e) with
              Edge (nt, _) -> nt
            | _ as t -> raise (Failure ("Expected Edge, received " ^
string_of_typ t))
          ) graph_lit
        in
        let get_edge_types (graph_lit : expr list) =
          List.map (fun (e) -> match fst (expr e) with
              Edge (_, et) -> et
            | _ as t -> raise (Failure ("Expected Edge, received " ^
string_of_typ t))
          ) graph_lit
        in
        (* Verify node types are the same for each EdgeLit *)
        check_all_same_types (get_node_types l) "node type";
        (* Verify edge types are the same for each EdgeLit *)
        check_all_same_types (get_edge_types l) "edge type";
        let sl = List.map (fun (a) -> (expr a)) l in
        let nt = match fst (List.nth sl 0) with
            Edge (nt, _) -> nt
          | _ as t -> raise (Failure ("Expected Edge, received " ^
string_of_typ t))
        in
        let et = match fst (List.nth sl 0) with
            Edge (_, et) -> et
          | _ as t -> raise (Failure ("Expected Edge, received " ^
string_of_typ t))
        in
        (* Return (Graph(t1, t2), SGraphLit sl) *)
        (Graph(nt, et), SGraphLit sl)
      | EdgeLit (node1, node2, edge) ->
        (* Verify node types are the same *)
        let (sn1, sn2, se) = (expr node1, expr node2, expr edge)
```

```
        in
        let verify_node_types = function
            (snode1, snode2, _) when (fst snode1) = (fst snode2) ->
()
          | (snode1, snode2, _) when (fst snode1) != (fst snode2) ->
              raise (Failure ("node types not the same: " ^
(string_of_typ (fst snode1)) ^ " and " ^ (string_of_typ (fst
snode2))))
          | _ -> raise (Failure ("this should not happen"))
        in
        verify_node_types (sn1, sn2, se);
        (* Return (Edge(t1, t2), SEdgeLit sl) *)
        (Edge(fst sn1, fst se), SEdgeLit (sn1, sn2, se))
      | ArrayLit l ->
          let get_array_types (array_lit : expr list) =
            List.map (fun (a) -> (fst (expr a))) array_lit
          in
          let check_typ = function
              Int -> ()
            | Double -> ()
            | Bool -> ()
            | Char -> ()
            | String -> ()
            | StructType(_) -> ()
            | Edge(_, _) -> ()
            | _ as t -> raise (Failure ("illegal array type: " ^
(string_of_typ t)))
          in
          List.iter check_typ (get_array_types l);
          check_all_same_types (get_array_types l) "array element
type";
          let a1 = List.nth (get_array_types l) 0 in
          let sl = List.map (fun (a) -> (expr a)) l in
          (Array(a1), SArrayLit sl)
      | True  -> (Bool, STrue)
      | False -> (Bool, SFalse)
      | Noexpr     -> (Int, SNoexpr)
      | Id s       -> (type_of_identifier s, SId s)
```

```
      | Index (s, e) -> (
        let (typ, se) = expr e in
        match (type_of_identifier s) with
          Array(t)    -> (match typ with
            Int -> (t, SArrayIndex(s, (typ, se)))
          | _ -> raise (Failure ("array index must be int, received "
^ (string_of_typ typ)))
          )
        | Map(kt, vt) -> (match typ with
            String | Int | Char | StructType(_) when typ = kt -> (vt,
SMapIndex(s, expr e))
          | _ -> raise (Failure ("map index must match key type,
received " ^ (string_of_typ typ) ^ " but expected " ^ (string_of_typ
kt)))
          )
        | _ as t -> raise (Failure ("illegal index operation: access
" ^ (string_of_typ (fst (expr e))) ^ " from " ^ (string_of_typ t)))
      )
      | StructFieldAccess (s, t) -> (
        (* verify struct field is a valid field of the struct *)
        match (type_of_identifier s) with
            StructType(r) -> (
                 let field_list = find_struct r in
                 let found = List.find (fun a -> fst a = t)
field_list in
                 ((snd found), SStructFieldAccess(s, t))
            )
          | _ -> raise (Failure ( s ^ " is not a struct type"))
      )
      | Assign(l, r) as ex ->
        let (lt, _) = expr l
        and (rt, _) = expr r in
        let err = "illegal assignment " ^ string_of_typ lt ^ " != " ^
          string_of_typ rt ^ " in " ^ string_of_expr ex in
        let check_for_inits = match lt with
            Map(a,b) ->
              let match_r = match r with
                  Call("map_init", []) ->
```

```
                    (Map(a,b), SAssign(expr l, (Map(a,b),
(SCall("map_init", [])))))
                | _ -> (check_assign lt rt err, SAssign(expr l, expr
r))
              in match_r
          | Graph(a,b) ->
              let match_r = match r with
                  Call("graph_init", []) ->
                    (Graph(a,b), SAssign(expr l, (Graph(a,b),
(SCall("graph_init", [])))))
                | _ -> (check_assign lt rt err, SAssign(expr l, expr
r))
              in match_r
          | Array(a) ->
              let match_r = match r with
                  Call("arr_init", []) ->
                    (Array(a), SAssign(expr l, (Array(a),
(SCall("arr_init", [])))))
                | _ -> (check_assign lt rt err, SAssign(expr l, expr
r))
              in match_r
          | _ -> (check_assign lt rt err, SAssign(expr l, expr r))
        in check_for_inits;
      | Reassign(l, r) as ex ->
        let (lt, _) = expr l
        and (rt, _) = expr r in
        let err = "illegal assignment " ^ string_of_typ lt ^ " != " ^
          string_of_typ rt ^ " in " ^ string_of_expr ex
        in (check_assign lt rt err, SAssign(expr l, expr r))
      | Unop(op, e) as ex ->
          let (t, e') = expr e in
          let ty = match op with
            Neg when t = Int || t = Double -> t
          | Not when t = Bool -> Bool
          | Incr when t = Int -> Int
          | Decr when t = Int -> Int
          | _ -> raise (Failure ("illegal unary operator " ^
                                 string_of_uop op ^ string_of_typ t ^
```

```
                                  " in " ^ string_of_expr ex))
          in (ty, SUnop(op, (t, e')))
      | Binop(e1, op, e2) as e ->
          let (t1, e1') = expr e1
          and (t2, e2') = expr e2 in
          (* All binary operators require operands of the same type
*)
          let same = t1 = t2 in
          (* Determine expression type based on operator and operand
types *)
          let ty = match op with
            Add | Sub | Mult | Div | Mod      when same && t1 = Int
-> Int
          | Add                               when same && t1 =
String -> String
          | DAdd | DSub | DMult | DDiv | DMod when same && t1 =
Double -> Double
          | Equal | Neq                       when same
-> Bool
          | Less | Leq | Greater | Geq
                              when same && (t1 = Int || t1 =
Double) -> Bool
          | And | Or                            when same && t1 =
Bool -> Bool
          | In when (
            match t2 with
              (* Array: right side is array and left side is array
element *)
              Array t1' -> t1' = t1
              (* Map: right side is map and left side is key *)
              | Map (kt, _) -> kt = t1
              (* Graph: right side is graph and left side is struct
(node) lit or edge lit *)
              | Graph (_nt, _et) -> (
                  match t1 with
                    StructType _ -> _nt = t1
                  | Edge (nt', et') -> nt' = _nt && et' = _et
                  | _ as t -> raise (Failure ("expected struct or
```

```
edge, received " ^ string_of_typ t))
                )
              | _ -> false
          )
          -> Bool
          | _ -> raise (
           Failure ("illegal binary operator " ^
                      string_of_typ t1 ^ " " ^ string_of_op op ^ " "
^
                      string_of_typ t2 ^ " in " ^ string_of_expr e))
          in (ty, SBinop((t1, e1'), op, (t2, e2')))
      | Call(fname, args) as call ->
          let fd = find_func fname in
          let param_length = List.length fd.formals in
          if List.length args != param_length then
            raise (Failure ("expecting " ^ string_of_int param_length
^
                            " arguments in " ^ string_of_expr call))
          else match fname with
            "lena" ->
              let (at, ae) = expr (List.nth args 0) in
              let at = match at with
                  Array(_) -> at
                | _ -> raise (Failure ("expecting array type,
received " ^ string_of_typ at))
              in
              (Int, SCall(fname, [(at, ae)]))
          | "lenm" ->
              let (mt, me) = expr (List.nth args 0) in
              let mt = match mt with
                  Map(_, _) -> mt
                | _ -> raise (Failure ("expecting map type, received
" ^ string_of_typ mt))
              in
              (Int, SCall(fname, [(mt, me)]))
          | "getKeys" ->
              let (mt, me) = expr (List.nth args 0) in
              let (kt, mt) = match mt with
```

```
                Map(kt, _) -> (kt, mt)
              | _ -> raise (Failure ("expecting map type, received
" ^ string_of_typ mt))
              in
              (Array(kt), SCall(fname, [(mt, me)]))
          | "getSrc" | "getDst" ->
              let (et, ee) = expr (List.nth args 0) in
              let (nt, et) = match et with
                  Edge(nt, _) -> (nt, et)
                | _ -> raise (Failure ("expecting edge type, received
" ^ string_of_typ et))
              in
              (nt, SCall(fname, [(et, ee)]))
          | "getVal" ->
              let (et, ee) = expr (List.nth args 0) in
              let (vt, et) = match et with
                  Edge(_, vt) -> (vt, et)
                | _ -> raise (Failure ("expecting edge type, received
" ^ string_of_typ et))
              in
              (vt, SCall(fname, [(et, ee)]))
          | "setSrc" | "setDst" ->
              let (et, ee) = expr (List.nth args 0) in
              let (nt, ne) = expr (List.nth args 1) in
              let verify_node_types = function
                  (Edge(nt', _), nt) when nt' = nt -> ()
                | (Edge(nt', _), nt) -> raise (Failure ("expecting
matching node types, received " ^ string_of_typ nt' ^ " and " ^
string_of_typ nt))
                | ((_ as t1), (_ as t2)) -> raise (Failure
("expecting edge and node types, received " ^ string_of_typ t1 ^ "
and " ^ string_of_typ t2))
              in
              verify_node_types (et, nt);
              (nt, SCall(fname, [(et, ee); (nt, ne)]))
          | "setVal" ->
              let (et, ee) = expr (List.nth args 0) in
              let (vt, ve) = expr (List.nth args 1) in
```

```
              let verify_val_types = function
                  (Edge(_, vt'), vt) when vt' = vt -> ()
                | (Edge(_, vt'), vt) -> raise (Failure ("expecting
matching value types, received " ^ string_of_typ vt' ^ " and " ^
string_of_typ vt))
                | ((_ as t1), (_ as t2)) -> raise (Failure
("expecting edge and edge value types, received " ^ string_of_typ t1
^ " and " ^ string_of_typ t2))
              in
              verify_val_types (et, vt);
              (vt, SCall(fname, [(et, ee); (vt, ve)]))
          | "append" ->
              let (at, ae) = expr (List.nth args 0) in
              let (vt, ve) = expr (List.nth args 1) in
              let verify_arr_type = match at with
                    Array(_) -> (match vt with
                        | Int -> ()
                        | Double -> ()
                        | Bool -> ()
                        | Char -> ()
                        | String -> ()
                        | StructType _ -> ()
                        | Edge (_, _) -> ()
                        | _ -> raise (Failure("incompatible type"))
                    )
                    | _ -> raise (Failure("can only append to
array"))
              in
              verify_arr_type;
              (vt, SCall(fname, [(at, ae); (vt, ve)]))
          | "addEdge" ->
              let (gt, ge) = expr (List.nth args 0) in
              let (et, ee) = expr (List.nth args 1) in
              let verify_edge_types = function
                  (Graph(nt, vt), Edge(nt', vt')) when nt = nt' && vt
= vt' -> ()
                | (Graph(nt, vt), Edge(nt', vt')) -> raise (Failure
("expecting matching node and value types, received " ^ string_of_typ
```

```
nt ^ ", " ^ string_of_typ nt' ^ ", " ^ string_of_typ vt ^ ", " ^
string_of_typ vt'))
                | ((_ as t1), (_ as t2)) -> raise (Failure
("expecting graph and edge types, received " ^ string_of_typ t1 ^ "
and " ^ string_of_typ t2))
              in
              verify_edge_types (gt, et);
              (et, SCall(fname, [(gt, ge); (et, ee)]))
          | "getEdges" ->
              let (gt, ge) = expr (List.nth args 0) in
              let et = match gt with
                  Graph(nt, vt) -> Array(Edge(nt, vt))
                | _ -> raise (Failure ("expecting graph type,
received " ^ string_of_typ gt))
              in
              (et, SCall(fname, [(gt, ge)]))
          | "getNodes" ->
              let (gt, ge) = expr (List.nth args 0) in
              let et = match gt with
                  Graph(nt, _) -> Array(nt)
                | _ -> raise (Failure ("expecting graph type,
received " ^ string_of_typ gt))
              in
              (et, SCall(fname, [(gt, ge)]))
          | _ ->
              let check_call (ft, _) e =
                let (et, e') = expr e in
                let err = "illegal argument found " ^ string_of_typ
et ^
                  " expected " ^ string_of_typ ft ^ " in " ^
string_of_expr e
                in (check_assign ft et err, e')
              in
              let args' = List.map2 check_call fd.formals args
              in (fd.typ, SCall(fname, args'))
    in

    let check_bool_expr e =
```

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

    (* Return a semantically-checked statement i.e. containing sexprs
*)
    let rec check_stmt = function
        Expr e -> SExpr (expr e)
      | If(p, b1, b2) -> SIf(check_bool_expr p, List.map check_stmt
b1, List.map check_stmt b2)
      | IfElseifs (p, b1, l, b3) -> SIfElseifs(check_bool_expr p,
List.map check_stmt b1, (List.map (fun(_, b2) -> check_bool_expr p,
List.map check_stmt b2) l), List.map check_stmt b3)
      | For(e1, e2, e3, st) ->
       SFor(expr e1, check_bool_expr e2, expr e3, List.map
check_stmt st)
      | While(p, s) -> SWhile(check_bool_expr p, List.map check_stmt
s)
      | Return e -> let (t, e') = expr e in
       if t = func.typ then SReturn (t, e')
       else raise (
       Failure ("return gives " ^ string_of_typ t ^ " expected " ^
             string_of_typ func.typ ^ " in " ^ string_of_expr e))

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

```
          in SBlock(check_stmt_list sl)
      | _ as s -> raise (Failure ("unexpected statement: " ^
(string_of_stmt s)))

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

codegen.ml

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

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

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

Detailed documentation on the OCaml LLVM library:

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

*)

module L = Llvm
module A = Ast
open Sast

module StringMap = Map.Make(String)

(* translate : Sast.program -> Llvm.module *)
```

```
let translate (globals, _, functions) =
  let context    = L.global_context () in
  let llmem = L.MemoryBuffer.of_file "gae.bc" in
  let llm = Llvm_bitreader.parse_bitcode context llmem in
  (* Create the LLVM compilation module into which
     we will generate code *)
  let the_module = L.create_module context "gae" 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 str_t      = L.pointer_type (L.i8_type context)
  and void_ptr_t = L.pointer_type (L.i8_type context)
  and arr_t      = L.pointer_type (match L.type_by_name llm
"struct.Array" with
      None -> raise (Failure "struct.Array isn't defined.")
    | Some x -> x)
  and arr_elmt_t = L.pointer_type (match L.type_by_name llm
"struct.Array_element" with
      None -> raise (Failure "struct.Array_element isn't defined.")
    | Some x -> x)
  and map_t      = L.pointer_type (match L.type_by_name llm
"struct.Map" with
      None -> raise (Failure "Missing implementation for struct Map")
    | Some t -> t)
  and struct_t   = L.pointer_type (match L.type_by_name llm
"struct.Struct" with
      None -> raise (Failure "Struct implementation not defined")
    | Some t -> t)
  and struct_element_t = L.pointer_type (match L.type_by_name llm
"struct.Struct_element" with
      None -> raise (Failure "Struct_element implementation not
defined")
    | Some t -> t)
  and edge_t     = L.pointer_type (match L.type_by_name llm
"struct.Edge" with
```

```
      None -> raise (Failure "Edge implementation not defined")
    | Some t -> t)
  and graph_t    = L.pointer_type (match L.type_by_name llm
"struct.Graph" with
      None -> raise (Failure "Graph implementation not defined")
    | Some t -> t)
  in
  (* TODO: Add rest of tpyes *)

  (* Return the LLVM type for a gae type *)
  let ltype_of_typ = function
      A.Int       -> i32_t
    | A.Bool      -> i1_t
    | A.Double    -> float_t
    | A.String    -> str_t
    | A.Array _   -> arr_t
    | A.Map _ -> map_t
    | A.StructType _ -> struct_t
    | A.Edge (_, _) -> edge_t
    | A.Graph (_, _) -> graph_t
    | _ -> raise (Failure "not implemented")
  in
  (* TODO: add rest of types *)

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

  (* print functions *)
  let printf_t : L.lltype = L.var_arg_function_type i32_t [|
```

```
L.pointer_type i8_t |] in
  let printf_func : L.llvalue = L.declare_function "printf" printf_t
the_module in

  (* misc function *)
  let int_of_double_t = L.function_type i32_t [| float_t |] in
  let int_of_double_f = L.declare_function "int_of_double"
int_of_double_t the_module in
  let double_of_int_t = L.function_type float_t [| i32_t |] in
  let double_of_int_f = L.declare_function "double_of_int"
double_of_int_t the_module in

  (* string functions *)
  let string_length_t = L.function_type i32_t [| str_t |] in
  let string_length_f = L.declare_function "string_length"
string_length_t the_module in
  let string_concat_t = L.function_type str_t [| str_t; str_t |] in
  let string_concat_f = L.declare_function "string_concat"
string_concat_t the_module in
  let string_equals_t = L.function_type i32_t [| str_t; str_t |] in
  let string_equals_f = L.declare_function "string_equals"
string_equals_t the_module in

  (* array functions *)
  let arr_init_t = L.function_type arr_t [||] in
  let arr_init_f = L.declare_function "arr_init" arr_init_t
the_module in
  let arr_set_contains_struct_t = L.function_type arr_t [| arr_t |]
in
  let arr_set_contains_struct_f = L.declare_function
"arr_set_contains_struct" arr_set_contains_struct_t the_module in
  let arr_append_t = L.function_type arr_elmt_t [| arr_t; void_ptr_t
|] in
  let arr_append_f = L.declare_function "arr_append" arr_append_t
the_module in
  let arr_get_t = L.function_type arr_elmt_t [| arr_t; i32_t |] in
  let arr_get_f = L.declare_function "arr_get" arr_get_t the_module
in
```

```
  let arr_set_t = L.function_type arr_elmt_t [| arr_t; void_ptr_t;
i32_t |] in
  let arr_set_f = L.declare_function "arr_set" arr_set_t the_module
in
  let arr_length_t = L.function_type i32_t [| arr_t |] in
  let arr_length_f = L.declare_function "arr_length" arr_length_t
the_module in
  let arr_contains_t = L.function_type i32_t [| arr_t; void_ptr_t |]
in
  let arr_contains_f = L.declare_function "arr_contains"
arr_contains_t the_module in

  (* map functions *)
  let map_init_t = L.function_type map_t [||] in
  let map_init_f = L.declare_function "map_init" map_init_t
the_module in
  let map_set_contains_struct_t = L.function_type map_t [| map_t |]
in
  let map_set_contains_struct_f = L.declare_function
"map_set_contains_struct" map_set_contains_struct_t the_module in
  let map_get_t = L.function_type void_ptr_t [| map_t; void_ptr_t |]
in
  let map_get_f = L.declare_function "map_get" map_get_t the_module
in
  let map_set_t = L.function_type void_ptr_t [| map_t; void_ptr_t;
void_ptr_t |] in
  let map_set_f = L.declare_function "map_set" map_set_t the_module
in
  let map_length_t = L.function_type i32_t [| map_t |] in
  let map_length_f = L.declare_function "map_length" map_length_t
the_module in
  let map_contains_t = L.function_type i32_t [| map_t; void_ptr_t |]
in
  let map_contains_f = L.declare_function "map_contains"
map_contains_t the_module in
  let map_get_keys_t = L.function_type arr_t [| map_t |] in
  let map_get_keys_f = L.declare_function "map_get_keys"
map_get_keys_t the_module in
```

```
  (* struct functions *)
  let struct_init_t = L.function_type struct_t [||] in
  let struct_init_f = L.declare_function "struct_init" struct_init_t
the_module in
  let struct_add_t = L.function_type struct_element_t [| struct_t;
str_t; void_ptr_t |] in
  let struct_add_f = L.declare_function "struct_add" struct_add_t
the_module in
  let struct_get_t = L.function_type void_ptr_t [| struct_t; str_t |]
in
  let struct_get_f = L.declare_function "struct_get" struct_get_t
the_module in
  let struct_set_t = L.function_type struct_element_t [| struct_t;
str_t; void_ptr_t |] in
  let struct_set_f = L.declare_function "struct_set" struct_set_t
the_module in
  let struct_equals_t = L.function_type i32_t [| struct_t; struct_t
|] in
  let struct_equals_f = L.declare_function "struct_equals"
struct_equals_t the_module in

  (* edge functions *)
  let edge_init_t = L.function_type edge_t [| struct_t; struct_t;
struct_t |] in
  let edge_init_f = L.declare_function "edge_init" edge_init_t
the_module in
  let edge_get_src_t = L.function_type struct_t [| edge_t |] in
  let edge_get_src_f = L.declare_function "edge_get_src"
edge_get_src_t the_module in
  let edge_get_dst_t = L.function_type struct_t [| edge_t |] in
  let edge_get_dst_f = L.declare_function "edge_get_dst"
edge_get_dst_t the_module in
  let edge_get_val_t = L.function_type struct_t [| edge_t |] in
  let edge_get_val_f = L.declare_function "edge_get_val"
edge_get_val_t the_module in
  let edge_set_src_t = L.function_type struct_t [| edge_t; struct_t
|] in
```

```
  let edge_set_src_f = L.declare_function "edge_set_src"
edge_set_src_t the_module in
  let edge_set_dst_t = L.function_type struct_t [| edge_t; struct_t
|] in
  let edge_set_dst_f = L.declare_function "edge_set_dst"
edge_set_dst_t the_module in
  let edge_set_val_t = L.function_type struct_t [| edge_t; struct_t
|] in
  let edge_set_val_f = L.declare_function "edge_set_val"
edge_set_val_t the_module in

  (* graph functions *)
  let graph_init_t = L.function_type graph_t [||] in
  let graph_init_f = L.declare_function "graph_init" graph_init_t
the_module in
  let graph_add_edge_t = L.function_type edge_t [| graph_t; edge_t |]
in
  let graph_add_edge_f = L.declare_function "graph_add_edge"
graph_add_edge_t the_module in
  let graph_get_edges_t = L.function_type arr_t [| graph_t |] in
  let graph_get_edges_f = L.declare_function "graph_get_edges"
graph_get_edges_t the_module in
  let graph_get_nodes_t = L.function_type arr_t [| graph_t |] in
  let graph_get_nodes_f = L.declare_function "graph_get_nodes"
graph_get_nodes_t the_module in
  let graph_contains_node_t = L.function_type i32_t [| graph_t;
struct_t |] in
  let graph_contains_node_f = L.declare_function
"graph_contains_node" graph_contains_node_t the_module in

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

```
      in
      let ftype = L.function_type (ltype_of_typ fdecl.styp)
formal_types in
      StringMap.add name (L.define_function name ftype the_module,
fdecl) m
    in
    List.fold_left function_decl StringMap.empty functions
  in

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

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

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

```
            StringMap.add n local m

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

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

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

    (* Construct code for an expression; return its value *)
    let rec expr builder ((styp, e) : sexpr) = match e with
           SIntLit i   -> L.const_int i32_t i
      | SDubLit l   -> L.const_float float_t l
      | SStrLit s   -> L.build_global_stringptr s "str" builder
      | SCharLit c  -> L.build_global_stringptr c "char" builder
      | STrue       -> L.const_int i1_t 1
      | SFalse      -> L.const_int i1_t 0
      | SArrayLit contents -> let rec arr_fill arr = (function
              [] -> arr
            | sx :: rest ->
                let (typ, _) = sx in
                let data = (match typ with
                  | _ -> let data = L.build_malloc (ltype_of_typ typ)
"data" builder in
```

```
                    let llvalue = expr builder sx in
                    ignore (L.build_store llvalue data builder);
data)
                in
                let data = L.build_bitcast data void_ptr_t "data"
builder in
                ignore (L.build_call arr_append_f [| arr; data |]
"arr_append" builder);
                arr_fill arr rest)
            in
            let el_typ = fst (List.nth contents 0) in
            let arr = match el_typ with
                A.StructType(_) ->
                  let arr = L.build_call arr_init_f [||] "arr_init"
builder in
                  L.build_call arr_set_contains_struct_f [| arr |]
"arr_set_contains_struct" builder
              | _ ->
                  L.build_call arr_init_f [||] "arr_init" builder
            in
            ignore(arr_fill arr (List.rev contents)); arr
      | SNoexpr     -> L.const_int i32_t 0
      | SStructLit contents -> let rec struct_fill sstruct =
(function
              [] -> sstruct
            | (sfield, svalue) :: rest ->
              let field = L.build_global_stringptr sfield "str"
builder in
              let (vtyp, _) = svalue in
              let value = (match vtyp with
                | _ -> let value = L.build_malloc (ltype_of_typ vtyp)
"value" builder in
                  let llvalue = expr builder svalue
                  in ignore (L.build_store llvalue value builder);
value)
              in
              let value = L.build_bitcast value void_ptr_t "value"
builder in
```

```
              ignore (L.build_call struct_add_f [| sstruct; field;
value |] "struct_add" builder);
              struct_fill sstruct rest)
        in
        let sstruct = L.build_call struct_init_f [||] "struct_init"
builder in
        ignore(struct_fill sstruct contents); sstruct
      | SMapLit contents ->
        let rec map_fill map_init = function
              [] -> map_init
            | (skey, svalue) :: rest ->
              let (ktyp, _) = skey in
              let key_ = (
                let key_ = L.build_malloc (ltype_of_typ ktyp) "key"
builder in
                let llkey = expr builder skey in
                ignore (L.build_store llkey key_ builder); key_
              ) in
              let key = L.build_bitcast key_ void_ptr_t "key" builder
in
              let (vtyp, _) = svalue in
              let value_ = (
                let data = L.build_malloc (ltype_of_typ vtyp) "val"
builder in
                let llvalue = expr builder svalue in
                ignore (L.build_store llvalue data builder); data
              ) in
              let value = L.build_bitcast value_ void_ptr_t "val"
builder in
              ignore (L.build_call map_set_f [| map_init; key; value
|] "map_set" builder);
              map_fill map_init rest
        in
        let key_typ = fst(fst (List.nth contents 0)) in
        let map = match key_typ with
            A.StructType(_) ->
              let map = L.build_call map_init_f [||] "map_init"
builder in
```

```
            L.build_call map_set_contains_struct_f [| map |]
"map_set_contains_struct" builder
          | _ ->
            L.build_call map_init_f [||] "map_init" builder
        in
        ignore(map_fill map contents); map
      | SEdgeLit (src, dst, value) ->
        let src' = expr builder src
        and dst' = expr builder dst
        and val' = expr builder value in
          L.build_call edge_init_f [| src'; dst'; val' |] "edge_init"
builder
      | SGraphLit contents ->
        let rec g_fill g = (function
          [] -> g
        | sx :: rest ->
            let edge = expr builder sx in
            ignore (L.build_call graph_add_edge_f [| g; edge |]
"graph_add_edge" builder);
            g_fill g rest)
        in
        let g = L.build_call graph_init_f [||] "graph_init" builder
in
        ignore(g_fill g (List.rev contents)); g
      | SId s       -> L.build_load (lookup s) s builder
      | SAssign ((_, le), (rt, re)) -> (match le with
            SId (s) ->
              let re' = expr builder (rt, re) in
              ignore(L.build_store re' (lookup s) builder); re'
          | SArrayIndex (id, e) ->
              let ltype = ltype_of_typ styp in
              let arr = L.build_load (lookup id) id builder in
              let index = expr builder e in
              let data = L.build_malloc ltype "data" builder in
              ignore(L.build_store (expr builder (rt, re)) data
builder);
              let data = L.build_bitcast data void_ptr_t "data"
builder in
```

```
              ignore(L.build_call arr_set_f [| arr; data; index |]
"arr_set" builder);
              data
          | SMapIndex (s, e) ->
              let map = L.build_load (lookup s) "map" builder in
              let (kt, _) = e in
              let key_ = (
                let key_ = L.build_malloc (ltype_of_typ kt) "key"
builder in
                let llkey = expr builder e in
                ignore (L.build_store llkey key_ builder); key_
              ) in
              let key = L.build_bitcast key_ void_ptr_t "key" builder
in
              let data_ = (
                let data = L.build_malloc (ltype_of_typ rt) "val"
builder in
                let llvalue = expr builder (rt, re) in
                ignore (L.build_store llvalue data builder); data
              ) in
              let value = L.build_bitcast data_ void_ptr_t "val"
builder in
              ignore (L.build_call map_set_f [| map; key; value |]
"map_set" builder);
              value
          | SStructFieldAccess (s, t) ->
            let field = L.build_global_stringptr t "str" builder in
            let ltype = ltype_of_typ styp in
            let sstruct = L.build_load (lookup s) s builder in
            let value = L.build_malloc ltype "value" builder in
            ignore(L.build_store (expr builder (rt, re)) value
builder);
            let value = L.build_bitcast value void_ptr_t "value"
builder in
            ignore(L.build_call struct_set_f [| sstruct; field; value
|] "struct_set" builder);
            value
          | _ -> raise (Failure "not implemented")
```

```
      )
      | SReassign (_, _) -> raise (Failure "Reassign should never be
called in codegen")
      | SStructFieldAccess (s, t) ->
          let field = L.build_global_stringptr t "str" builder in
          let ltype = ltype_of_typ styp in
          let sstruct = L.build_load (lookup s) s builder in
          let data_ptr = L.build_call struct_get_f [| sstruct; field
|] "struct_get" builder in
          let data_ptr = L.build_bitcast data_ptr (L.pointer_type
ltype) "value" builder in
          L.build_load data_ptr "value" builder
      | SMapIndex (s, e) -> (match e with
          (_ as kt, _) ->
            let ltype = ltype_of_typ kt in
            let key_ = (
              let key_ = L.build_malloc ltype "key" builder in
              let llkey = expr builder e in
              ignore (L.build_store llkey key_ builder); key_
            ) in
            let key = L.build_bitcast key_ void_ptr_t "key" builder
in
            let map = L.build_load (lookup s) "map" builder in
            let ptr = L.build_call map_get_f [| map; key |] "map_get"
builder in
            let ptr = L.build_bitcast ptr (L.pointer_type
(ltype_of_typ styp)) "data" builder in
            L.build_load ptr "data" builder
        )
      | SArrayIndex (id, e) ->
            let ltype = ltype_of_typ styp in
            let arr = L.build_load (lookup id) id builder in
            let index = expr builder e in
            let data_ptr = L.build_call arr_get_f [| arr; index |]
"arr_get" builder in
            let data_ptr = L.build_bitcast data_ptr (L.pointer_type
ltype) "data" builder in
            L.build_load data_ptr "data" builder
```

```
      | SBinop ((A.Double,_ ) as e1, op, e2) ->
            let e1' = expr builder e1
            and e2' = expr builder e2 in
            (match op with
              A.DAdd     -> L.build_fadd
            | A.DSub     -> L.build_fsub
            | A.DMult    -> L.build_fmul
            | A.DDiv     -> L.build_fdiv
            | A.Equal    -> L.build_fcmp L.Fcmp.Oeq
            | A.Neq      -> L.build_fcmp L.Fcmp.One
            | A.Less     -> L.build_fcmp L.Fcmp.Olt
            | A.Leq      -> L.build_fcmp L.Fcmp.Ole
            | A.Greater -> L.build_fcmp L.Fcmp.Ogt
            | A.Geq      -> L.build_fcmp L.Fcmp.Oge
            | A.And | A.Or ->
                raise (Failure "internal error: semant should have
rejected and/or on float")
          | _ -> raise (Failure "not implemented")
            ) e1' e2' "tmp" builder
      | SBinop ((A.String, _) as e1, op, e2) when op = A.Add || op =
A.Equal || op = A.Neq ->
          let e1' = expr builder e1
          and e2' = expr builder e2 in
          (match op with
            A.Add     -> L.build_call string_concat_f [| e1'; e2' |]
"string_concat" builder
          | A.Equal    -> (L.build_icmp L.Icmp.Eq) (L.const_int i32_t
0)
                (L.build_call string_equals_f [| e1'; e2' |]
"string_equals" builder) "tmp" builder
          | A.Neq      -> (L.build_icmp L.Icmp.Ne) (L.const_int i32_t
0)
                (L.build_call string_equals_f [| e1'; e2' |]
"string_equals" builder) "tmp" builder
          | _ -> raise (Failure "not implemented")
          )
      | SBinop ((t1, e1), op, (t2, e2)) when
        (op = A.Equal || op = A.Neq) &&
```

```
        (match (t1, t2) with
            (A.StructType(t1'), A.StructType(t2')) -> t1' = t2'
          | _ -> false
        ) ->
          let e1' = expr builder (t1, e1)
          and e2' = expr builder (t2, e2) in
          (match op with
            A.Equal   -> (L.build_icmp L.Icmp.Eq) (L.const_int i32_t
1)
                (L.build_call struct_equals_f [| e1'; e2' |]
"struct_equals" builder) "tmp" builder
          | A.Neq     -> (L.build_icmp L.Icmp.Ne) (L.const_int i32_t
1)
                (L.build_call struct_equals_f [| e1'; e2' |]
"struct_equals" builder) "tmp" builder
          | _ -> raise (Failure "not implemented")
          )
      | SBinop ((t1, e1), op, (t2, e2)) when op = A.In ->
        let e1' = expr builder (t1, e1)
        and e2' = expr builder (t2, e2) in
        let t1' = ltype_of_typ t1 in
        let val_ = L.build_malloc t1' "val" builder in
        ignore (L.build_store e1' val_ builder);
        let value = L.build_bitcast val_ void_ptr_t "val" builder in
        (match t2 with
            A.Array(_) ->
              L.build_call arr_contains_f [| e2'; value |]
"arr_contains" builder
          | A.Map(_, _) ->
              L.build_call map_contains_f [| e2'; value |]
"map_contains" builder
          | A.Graph(_, _) ->
              (match t1 with
                  A.StructType(_) ->
                    let value = L.build_bitcast value struct_t
"struct" builder in
                    (L.build_icmp L.Icmp.Ne) (L.const_int i32_t 1)
(L.build_call graph_contains_node_f [| e2'; value |]
```

```
"graph_contains_node" builder) "tmp" builder
                (* | Edge(_, _) -> *)
                | _ -> raise (Failure ("expected edge or node,
received " ^ A.string_of_typ t1))
              )
          | _ -> raise (Failure "not implemented")
        )
      | SBinop ((A.Int, _) as e1, op, e2) ->
            let e1' = expr builder e1
            and e2' = expr builder e2 in
            (match op with
              A.Add     -> L.build_add
            | A.Sub     -> L.build_sub
            | A.Mult    -> L.build_mul
          | A.Div     -> L.build_sdiv
          | A.Mod     -> L.build_srem
            | A.And     -> L.build_and
            | A.Or      -> L.build_or
            | A.Equal   -> L.build_icmp L.Icmp.Eq
            | A.Neq     -> L.build_icmp L.Icmp.Ne
            | A.Less    -> L.build_icmp L.Icmp.Slt
            | A.Leq     -> L.build_icmp L.Icmp.Sle
            | A.Greater -> L.build_icmp L.Icmp.Sgt
            | A.Geq     -> L.build_icmp L.Icmp.Sge
          | _ -> raise (Failure "not implemented")
            ) e1' e2' "tmp" builder
      | SBinop ((A.Bool, _) as e1, op, e2) ->
            let e1' = expr builder e1
            and e2' = expr builder e2 in
            (match op with
           A.And -> L.build_and
          | A.Or -> L.build_or
          | _ -> raise (Failure "not implemented")
          ) e1' e2' "tmp" builder
      | SBinop (_,_,_) as e -> raise (Failure ("binop not implemented
for this expr " ^ string_of_sexpr (styp, e)))
      | SUnop(op, ((t, _) as e)) ->
          let e' = expr builder e in
```

```
            (match op with
              A.Neg when t = A.Double -> L.build_fneg e' "tmp"
builder
            | A.Neg                   -> L.build_neg e' "tmp"
builder
          | A.Not                     -> L.build_not e' "tmp" builder
          | A.Incr -> raise (Failure "Incr should never be called in
codegen")
          | A.Decr -> raise (Failure "Decr should never be called in
codegen")
          )
      | SCall ("printi", [e]) -> L.build_call printf_func [|
int_format_str ; (expr builder e) |] "printf" builder
      | SCall ("printb", [e]) -> L.build_call printf_func [|
bool_format_str ; (expr builder e) |] "printf" builder
      | SCall ("prints", [e]) -> L.build_call printf_func [|
string_format_str ; (expr builder e) |] "printf" builder
      | SCall ("printc", [e]) -> L.build_call printf_func [|
char_format_str ; (expr builder e) |] "printf" builder
      | SCall ("printd", [e]) -> L.build_call printf_func [|
float_format_str ; (expr builder e) |] "printf" builder
      | SCall ("lena", [a]) -> L.build_call arr_length_f [| expr
builder a |] "lena" builder
      | SCall ("lenm", [m]) -> L.build_call map_length_f [| expr
builder m |] "lenm" builder
      | SCall ("lens", [s]) -> L.build_call string_length_f [| expr
builder s |] "lens" builder
      | SCall ("getKeys", [m]) -> L.build_call map_get_keys_f [| expr
builder m |] "map_get_keys" builder
      | SCall ("map_init", []) ->
        (* L.build_call map_init_f [||] "map_init" builder *)
        (match styp with
            A.Map((_ as s), _) when (match s with A.StructType(_) ->
true | _ -> false)->
              let map = L.build_call map_init_f [||] "map_init"
builder in
              L.build_call map_set_contains_struct_f [| map |]
"map_set_contains_struct" builder
```

```
          | _ -> L.build_call map_init_f [||] "map_init" builder
        )
      | SCall ("graph_init", []) -> L.build_call graph_init_f [||]
"graph_init" builder
      | SCall ("arr_init", []) ->
        (* L.build_call arr_init_f [||] "arr_init" builder *)
        (match styp with
            A.Array(_ as s) when (match s with A.StructType(_) ->
true | _ -> false)->
              let arr = L.build_call arr_init_f [||] "arr_init"
builder in
              L.build_call arr_set_contains_struct_f [| arr |]
"arr_set_contains_struct" builder
          | _ -> L.build_call arr_init_f [||] "arr_init" builder
        )
      | SCall ("getSrc", [e]) -> L.build_call edge_get_src_f [| expr
builder e |] "edge_get_src" builder
      | SCall ("getDst", [e]) -> L.build_call edge_get_dst_f [| expr
builder e |] "edge_get_dst" builder
      | SCall ("getVal", [e]) -> L.build_call edge_get_val_f [| expr
builder e |] "edge_get_val" builder
      | SCall ("setSrc", [e; n]) -> L.build_call edge_set_src_f [|
expr builder e; expr builder n |] "edge_set_src" builder
      | SCall ("setDst", [e; n]) -> L.build_call edge_set_dst_f [|
expr builder e; expr builder n |] "edge_set_dst" builder
      | SCall ("setVal", [e; v]) -> L.build_call edge_set_val_f [|
expr builder e; expr builder v |] "edge_set_val" builder
      | SCall ("append", [a; e]) ->
        let (typ, _) = e in
        let data = L.build_malloc (ltype_of_typ typ) "data" builder
in
        ignore (L.build_store (expr builder e) data builder);
        let data = L.build_bitcast data void_ptr_t "data" builder in
        L.build_call arr_append_f [| expr builder a; data |]
"arr_append" builder
      | SCall ("addEdge", [g; e]) -> L.build_call graph_add_edge_f [|
expr builder g; expr builder e |] "graph_add_edge" builder
      | SCall ("getEdges", [g]) -> L.build_call graph_get_edges_f [|
```

```
expr builder g |] "graph_get_edges" builder
      | SCall ("getNodes", [g]) -> L.build_call graph_get_nodes_f [|
expr builder g |] "graph_get_nodes" builder
      | SCall ("int_of_double", [e]) -> L.build_call int_of_double_f
[| expr builder e |] "int_of_double" builder
      | SCall ("double_of_int", [e]) -> L.build_call double_of_int_f
[| expr builder e |] "double_of_int" builder
      | SCall (f, args) ->
        let (fdef, fdecl) = StringMap.find f function_decls in
        let llargs = List.rev (List.map (expr builder) (List.rev
args)) in
            let result = (match fdecl.styp with
             | _ -> f ^ "_result") in
        L.build_call fdef (Array.of_list llargs) result builder
    in

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

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

    let rec stmt builder = function
        SBlock sl -> List.fold_left stmt builder sl
      | SExpr e -> ignore(expr builder e); builder
      | SReturn e -> ignore(L.build_ret (expr builder e) builder );
```

```
                    builder
      | SIf (predicate, then_block, else_block) ->
         let bool_val = expr builder predicate in
            let merge_bb = L.append_block context "merge"
the_function in
         let build_br_merge = L.build_br merge_bb in (* partial
function *)

         let then_bb = L.append_block context "then" the_function in
         let then_builder = (List.map (fun (then_stmt) -> stmt
(L.builder_at_end context then_bb) then_stmt) (List.rev then_block))
in
         ignore((List.map (fun (then_stmt) -> add_terminal then_stmt
build_br_merge) then_builder));

            let else_bb = L.append_block context "else" the_function
in
         let else_builder = (List.map (fun (else_stmt) -> stmt
(L.builder_at_end context else_bb) else_stmt) (List.rev else_block))
in
         ignore((List.map (fun (else_stmt) -> add_terminal else_stmt
build_br_merge) else_builder));

            ignore(L.build_cond_br bool_val then_bb else_bb builder);
            L.builder_at_end context merge_bb
      | SIfElseifs _ -> raise (Failure "elseifs not defined.")
      | SBreak -> raise (Failure "break not defined")
      | SContinue -> raise (Failure "continue not defined")
      | SWhile (predicate, body) ->
        let pred_bb = L.append_block context "while" the_function in
        ignore(L.build_br pred_bb builder);

        let body_bb = L.append_block context "while_body"
the_function in
         let while_builder = (List.map (fun (while_stmt) -> stmt
(L.builder_at_end context body_bb) while_stmt) (List.rev body)) in
         ignore((List.map (fun (while_stmt) -> add_terminal
while_stmt (L.build_br pred_bb)) while_builder));
```

```
        let pred_builder = L.builder_at_end context pred_bb in
        let bool_val = expr pred_builder predicate in

        let merge_bb = L.append_block context "merge" the_function
in
        ignore(L.build_cond_br bool_val body_bb merge_bb
pred_builder);
        L.builder_at_end context merge_bb
      | SFor (e1, e2, e3, body) -> stmt builder
            ( SBlock [SExpr e1 ; SWhile (e2, [SBlock ((List.rev
body) @ [SExpr e3])]) ] )
    in

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

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

  List.iter build_function_body functions;
  the_module
```

gae.ml

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

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: ./gae.native [-a|-s] [file.gae]" in
  let channel = ref stdin in
  Arg.parse speclist (fun filename -> channel := open_in filename)
usage_msg;

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

gae.c

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

/*
 ============================================================
                    TYPEDEFS
 ============================================================
```

```
*/

typedef struct Array_element {
    struct Array_element *next;
    struct Array_element *prev;
    void *data;
} array_element;

typedef struct Array {
    int32_t length;
    array_element *head;
    array_element *tail;
    int32_t contains_struct;
} array;

typedef struct Map_element {
    struct Map_element *next;
    struct Map_element *prev;
    void *key;
    void *value;
} map_element;

typedef struct Map {
    int32_t length;
    map_element *head;
    map_element *tail;
    int32_t contains_struct;
    int32_t is_nodepair_map;
} map;

typedef struct Struct_element {
     struct Struct_element *next;
     struct Struct_element *prev;
     char *field;
     void *value;
} struct_element;

typedef struct Struct {
```

```
    struct_element *head;
    struct_element *tail;
  int32_t length;
 } sstruct;

 typedef struct Edge {
   sstruct *src;
   sstruct *dst;
   sstruct *value;
 } edge;

 typedef struct Graph {
   map *m;
   array *node_list;
 } graph;

 typedef struct NodePair {
   sstruct *src;
   sstruct *dst;
 } nodepair;

/*
  ======================================================
                  FUNCTION DECLARATIONS
  ======================================================
  */

array* arr_init();
array* arr_set_contains_struct(array *a);
array_element* arr_append(array *l, void *data);
void* arr_get(array *l, int index);
array_element* arr_set(array *l, void *data, int index);
int32_t arr_contains(array *l, void *data);
int32_t arr_length(array *l);

char *string_concat(char *s1, char *s2);
int string_equals(char *s1, char *s2);
int string_length(char *s);
```

```
map* map_init();
map* map_set_contains_struct(map *m);
map_element* map_get_el(map *m, void *key);
void* map_get(map *m, void *key);
void* map_set(map *m, void *key, void* value);
int32_t map_contains(map *m, void *key);
array* map_get_keys(map *m);
int32_t map_length(map *m);

sstruct * struct_init();
struct_element * struct_add(sstruct *s, char *field, void *value);
void* struct_get(sstruct *s, char *field);
struct_element* struct_set(sstruct *s, char *field, void *value);
int32_t struct_equals_helper(sstruct *s1, sstruct *s2);
int32_t struct_equals(sstruct *s1, sstruct *s2);

edge *edge_init(sstruct *src, sstruct *dst, sstruct *value);
sstruct *edge_get_src(edge *e);
sstruct *edge_get_dst(edge *e);
sstruct *edge_get_val(edge *e);
sstruct *edge_set_src(edge *e, sstruct *src);
sstruct *edge_set_dst(edge *e, sstruct *dst);
sstruct *edge_set_val(edge *e, sstruct *val);

int32_t nodepair_equals(nodepair *a, nodepair *b);
graph *graph_init();
edge *graph_add_edge(graph *g, edge *e);
array *graph_get_edges(graph *g);
array *graph_get_nodes(graph *g);
int32_t graph_contains_node(graph *g, sstruct *n);

double double_of_int(int x);
int int_of_double(double x);

/*
 ======================================================
                    ARRAY FUNCTIONS
```

```
 ==========================================================
 */

 //init array
 array* arr_init() {
     array* l = (array *) malloc(sizeof(array));
     l->length = 0;
     l->head = NULL;
     l->tail = NULL;
     l->contains_struct = 0;
     return l;
 }

// call this if the array contains a struct
array* arr_set_contains_struct(array *a) {
  a->contains_struct = 1;
  return a;
}

 //append element: can be any type
 array_element* arr_append(array *l, void *data) {
     array_element *le = (array_element*)
malloc(sizeof(array_element));
     le->data = data;
     le->next = NULL;
     le->prev = l->tail;
     if(l->tail) {
         l->tail->next = le;
     }
     else { //empty array
         l->head = le;
     }
     l->tail = le;
     l->length += 1;

     return le;
 }
```

```
//access element
void* arr_get(array *l, int index) {
    if(index >= l->length) {
        return NULL;
    }
    array_element *curr = l->head;
    int count = 0;
    while(count < index) {
        curr = curr->next;
        count++;
    }
    return curr->data;
}

//set element
array_element* arr_set(array *l, void *data, int index) {
    if(index >= l->length) {
        return NULL;
    }
    array_element *curr = l->head;
    int count = 0;
    while(count < index) {
        curr = curr->next;
        count++;
    }
    free(curr->data);
    curr->data = data;
    return curr;
}

int32_t arr_contains(array *l, void *data) {
  array_element *curr = l->head;
  while(curr != NULL) {
    if (
      (
        (
          data == curr->data ||
          strcmp(data, curr->data) == 0
```

```
        ) && !l->contains_struct
      ) ||
      (l->contains_struct && struct_equals(*(sstruct **)data,
*(sstruct **)curr->data))
    ) {
      return 1;
    }
    curr = curr->next;
  }
  return 0;

}

int32_t arr_length(array *l) {
    return l->length;
}

/*
============================================================
                    STRING FUNCTIONS
============================================================
*/

char *string_concat(char *s1, char *s2) {
    char *new = (char *) malloc(strlen(s1) + strlen(s2) + 1);
    strcpy(new, s1);
    strcat(new, s2);
    return new;
}

int string_equals(char *s1, char *s2) {
    return !(strcmp(s1, s2) == 0);
}

int string_length(char *s) {
    return strlen(s);
}
```

```
/*
=======================================================
                    MAP FUNCTIONS
=======================================================
*/

/* Initialize map  */
map* map_init() {
  map* m = (map *) malloc(sizeof(map));
  m->length = 0;
  m->head = NULL;
  m->tail = NULL;
  m->contains_struct = 0;
  return m;
}

map* map_set_contains_struct(map *m) {
  m->contains_struct = 1;
  return m;
}

/* Get map element with key, used as a helper function */
map_element* map_get_el(map *m, void *key) {
  map_element *curr = m->head;
  int count = 0;
  while (count < m->length) {
    if (
      (
        (
          *(int *)curr->key == *(int *)key ||   /* if int key */
          *(char *)curr->key == *(char *)key || /* if char key */
          strcmp(curr->key, key) == 0           /* if str key */
        ) && !m->contains_struct && !m->is_nodepair_map
      ) ||
      (m->contains_struct && struct_equals(*(sstruct **)key,
*(sstruct **)curr->key)) || /* if struct key */
      (m->is_nodepair_map && nodepair_equals((nodepair *)key,
(nodepair *)curr->key))    /* if nodepair key */
```

```
    ) {
      break;
    }
    curr = curr->next;
    count++;
  }
  if (count == m->length) {
    // Not found
    return NULL;
  } else {
    // Found, return value
    return curr;
  }
}

/* Get value with int key */
void* map_get(map *m, void *key) {
  map_element *element = map_get_el(m, key);
  if (element == NULL) {
    return 0;
  } else {
    return element->value;
  }
}

/* Set value with int key */
void* map_set(map *m, void *key, void* value) {
  map_element *element = map_get_el(m, key);
  if (element == NULL) {
    // create new element and return
    map_element *new_element = malloc(sizeof(map_element));
    // printf("Malloc: at %p\n", new_element);
    // init new element
    new_element->key = key;
    new_element->value = value;
    new_element->prev = m->tail;
    new_element->next = NULL;
    // update pointers
```

```
    if (m->tail) {
      m->tail->next = new_element;
    } else {
      m->head = new_element;
    }
    m->tail = new_element;
    // increment
    m->length++;
  } else {
    // overwrite existing value
    element->value = value;
  }
     return value;
}

int32_t map_contains(map *m, void *key) {
  map_element *curr = m->head;
  while(curr != NULL) {
      if (
        (
          (
            *(int *)curr->key == *(int *)key ||   /* if int key */
            *(char *)curr->key == *(char *)key || /* if char key */
            strcmp(curr->key, key) == 0           /* if str key */
          ) && !m->contains_struct && !m->is_nodepair_map
        ) ||
        (m->contains_struct && struct_equals(*(sstruct **)key,
*(sstruct **)curr->key)) || /* if struct key */
        (m->is_nodepair_map && nodepair_equals((nodepair *)key,
(nodepair *)curr->key))    /* if nodepair key */
      ) {
        return 1;
      }
      curr = curr->next;
  }
  return 0;
}
```

```
array* map_get_keys(map *m) {
  array *a = arr_init();
  map_element *curr = m->head;
  while (curr != NULL) {
    arr_append(a, curr->key);
    curr = curr->next;
  }
  return a;
}

int32_t map_length(map *m) {
  return m->length;
}

/*
================================================================================
                Struct Functions
================================================================================
*/

sstruct * struct_init() {
     sstruct* s = (sstruct *) malloc(sizeof(sstruct));
     s->length = 0;
     s->head = NULL;
     s->tail = NULL;
     return s;
}

struct_element * struct_add(sstruct *s, char *field, void *value) {
     struct_element *s1 = (struct_element *)
malloc(sizeof(struct_element));
     s1->field = field;
     s1->value = value;
     s1->prev = s->tail;
     if(s->tail){
          s->tail->next = s1;
```

```
	}
	else{
		s->head = s1;
	}
	s->tail = s1;
	s->length += 1;

	return s1;
}

void* struct_get(sstruct *s, char *field) {
	struct_element *cur = s->head;
	int count = 0;
	while(count < s->length){
		if (strcmp(cur->field, field) == 0){
			break;
		}
		else{
			cur = cur->next;
			count ++;
		}
	}
	return cur->value;
}

struct_element* struct_set(sstruct *s, char *field, void *value) {
	struct_element *cur = s->head;
	int count = 0;
	while(count < s->length){
		if (strcmp(cur->field, field) == 0){
			break;
		}
		else{
			cur = cur->next;
			count ++;
		}
	}
	free(cur->value);
```

```
      cur->value = value;
      return cur;
}

int32_t struct_equals_helper(sstruct *s1, sstruct *s2) {
  /* Since fields are potentially out of order, we'd need to validate
each one. */
  struct_element *cur1 = s1->head;
  struct_element *cur2 = s2->head;
  while (cur1 != NULL) {
    while (cur2 != NULL && strcmp(cur1->field, cur2->field) != 0) {
      cur2 = cur2->next;
    }
    // if field is not found in s2
    if (cur2 == NULL) {
      return 0;
    }
    // now we know we're on the same field
    if (
      *(int *)cur1->value == *(int *)cur2->value || /* int */
      *(char *)cur1->value == *(char *)cur2->value || /* char */
      strcmp(cur1->value, cur2->value) == 0 /* string */
    ) {
      cur1 = cur1->next;
      continue;
    }
    // now we know fields don't equal
    return 0;
  }
  return 1;
}

int32_t struct_equals(sstruct *s1, sstruct *s2) {
  /* Sanity check on length */
  if (s1->length != s2->length) {
    return 0;
  }
  return struct_equals_helper(s1, s2) || struct_equals_helper(s2,
```

```
s1);
}

/*
=======================================================
                    EDGE FUNCTIONS
=======================================================
*/

edge *edge_init(sstruct *src, sstruct *dst, sstruct *value) {
  edge *e = (edge *)malloc(sizeof(edge));
  e->src = src;
  e->dst = dst;
  e->value = value;
  return e;
}

sstruct *edge_get_src(edge *e) {
  return e->src;
}

sstruct *edge_get_dst(edge *e) {
  return e->dst;
}

sstruct *edge_get_val(edge *e) {
  return e->value;
}

sstruct *edge_set_src(edge *e, sstruct *src) {
  e->src = src;
  return src;
}

sstruct *edge_set_dst(edge *e, sstruct *dst) {
  e->dst = dst;
  return dst;
}
```

```
sstruct *edge_set_val(edge *e, sstruct *value) {
  e->value = value;
  return value;
}

/*
=====================================================
              GRAPH/NODEPAIR FUNCTIONS
=====================================================
*/

/*
  Implementation note: we use a map to store the graph, where the key
type is
  nodepair (an ordered pair of nodes), and value is sstruct (the edge
value),
  i.e. map<nodepair, sstruct>
*/

int32_t nodepair_equals(nodepair *a, nodepair *b) {
  return struct_equals(a->src, b->src) && struct_equals(a->dst,
b->dst);
}

graph *graph_init() {
  graph *g = malloc(sizeof(graph));
  g->m = map_init();
  g->m->is_nodepair_map = 1;
  g->node_list = arr_init();
  g->node_list->contains_struct = 1;
  return g;
}

/* Node: this function overwrites an edge if it already exists */
edge *graph_add_edge(graph *g, edge *e) {
  nodepair *np = malloc(sizeof(nodepair));
  np->src = e->src;
```

```
  np->dst = e->dst;
  map_set(g->m, np, e->value);
  /* add src to node list */
  if (!arr_contains(g->node_list, &np->src)) {
    sstruct **ptr = malloc(sizeof(sstruct *));
    *ptr = np->src;
    arr_append(g->node_list, ptr);
  }
  if (!arr_contains(g->node_list, &np->dst)) {
    sstruct **ptr = malloc(sizeof(sstruct *));
    *ptr = np->dst;
    arr_append(g->node_list, ptr);
  }
  return e;
}

array *graph_get_edges(graph *g) {
  array *a = arr_init();
  map_element *curr = g->m->head;
  while (curr != NULL) {
    edge *e = edge_init(((nodepair *)curr->key)->src, ((nodepair
*)curr->key)->dst, curr->value);
    edge **e_ptr = malloc(sizeof(edge *));
    *e_ptr = e;
    arr_append(a, e_ptr);
    curr = curr->next;
  }
  return a;
}

array *graph_get_nodes(graph *g) {
  return g->node_list;
}

int32_t graph_contains_node(graph *g, sstruct *n) {
  return arr_contains(g->node_list, n);
}
```

```
/*
=================================================
           Int to/from Double
=================================================
*/

double double_of_int(int x) {
      return ((double) x);
}

int int_of_double(double x) {
      return (int) x;
}
```

Makefile

```
.PHONY : all
all : gae.native gae.o

.PHONY : gae.native
gae.native :
      ocamlbuild -use-ocamlfind -pkgs
llvm,llvm.analysis,llvm.bitreader -cflags -w,+a-4 \
            gae.native
      gcc -c gae.c
      clang -emit-llvm -o gae.bc -c gae.c -Wno-varargs

.PHONY : clean
clean :
      rm -rf *.o *.bc *.out *.s *.ll *.native
```

compile.sh

```
#!/bin/bash
if [ "$#" -ne 1 ]; then
    echo "Illegal number of parameters"
    exit 1
```

```
fi

FILE=` echo $1 | cut -d/ -f2`

./gae.native $1 > "$FILE.ll"
llc $FILE.ll > $FILE.s
gcc -c gae.c
#clang -emit-llvm -o gae.bc -c gae.c -Wno-varargs
gcc -no-pie -o $FILE.out $FILE.s gae.o
```