Konig – Final Report

Matteo Sandrin (ms4911)

Delilah Beverly (db3250)

Lord Crawford (lrc2161)

Columbia University

April 26, 2021

Table of Contents

1. Introduction

Graphs have an important role in a number of applications including networks, data processing, databases and everything in between. The Konig programming language is aimed at making the creation and manipulation of graphs easier and more enjoyable.

Konig is an imperative, statically typed language. The language's syntax is similar to C, but with the addition of a number of operators and functions specific to graph theory. Furthermore, Konig uses a syntax for generic types similar to Java.

The language is named after the "Seven Bridges of Königsberg", a famous math problem that laid the foundations of graph theory. It also means "king" in German.

2. Konig Tutorial

2.1 Environment Setup

2.1.1 LLVM

First of all, before installing Konig, install the OCaml package manager (OPAM), the LLVM libraries and the LLVM bindings for OCaml. On MacOS, these steps can be accomplished by running the following commands in your terminal:

```
$> brew install opam
$> brew install llvm@11
$> opam init
$> opam install llvm
```

On Linux-based systems, you can use the following commands instead:

```
$> sudo apt install opam
$> sudo apt install llvm
$> sudo apt install ocamlbuild
$> sudo apt install clang
$> opam init
$> opam install llvm
```

2.1.2 Graphviz

In order to support the visualization of graphs as an image, Konig depends on the Graphviz library. However, the Graphviz library is not required to compile and run Konig programs. Konig will cut off the use of the `viz()` function if it's unable to find the Graphviz library, but all other functionality will be unaffected.

To install the Graphviz library on MacOS, run the following command in your terminal:

```
$> brew install graphviz
```

On Linux-based systems, you can use the following command instead:

```
$> sudo apt install graphviz libgraphviz-dev libcgraph6
```

On MacOS, the path to the Graphviz libraries is already configured in the build script. On Linux-based systems however, the path to the Graphviz libraries may vary between distributions, and should be changed in the build script. Simply edit the `GRAPHVIZ_PATH` variable within the `compile.sh` file with the correct path.

If for any reason you cannot install the Graphviz library, set the variable `USE_GRAPHVIZ` to `0` in `compile.sh`

The build script will check for the presence of the Graphviz libraries at compile time. If Graphviz is not present the compiler will show a warning message, but still compile and run programs successfully. If the user attempts to call the `viz()` function while the Graphviz library is not available, the call will fail gracefully with a warning message to the user and continue execution of the program.

2.2 Building Konig

To build the Konig compiler, first clone the Konig git repository with the command

```
$> git clone https://github.com/matteosandrin/konig
```

Then, build the Konig compiler by running the following commands, and the test suite will be run automatically after a successful compilation:

```
$> cd konig
```

```
$> make
```

The test suite for Konig can also be run independently with the following command:

```
$> make test
```

Note: Konig's test suite requires Python to be installed on the system.

2.3 Running a Program

After compiling Konig, we can start compiling programs written in Konig. The following is a sample program, which creates three nodes, adds them to a graph and connects all the nodes with each other. Finally, the program generates a visual representation of the graph and saves it as a PDF, shown beside the code.

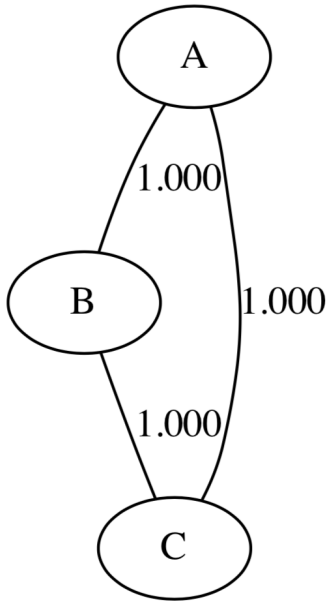
```
ko int main() {
    node<string> a;
    node<string> b;
    node<string> c;
    graph<string> g;

    g = new graph{};
    a = new node{"A"};
    b = new node{"B"};
    c = new node{"C"};

    a @ g;
    b @ g;
    c @ g;

    setEdge(g, a, b, 1.0);
    setEdge(g, b, c, 1.0);
    setEdge(g, c, a, 1.0);

    viz(g, "sample.pdf");
}
```



If the file above is named `sample.ko`, then compiling it is as simple as running:

```
$> ./compile.sh sample.ko
```

The resulting executable, sample.out, can then be directly executed:

```
$> ./sample.out
```

3. Language Reference Manual

3.1 Lexical Conventions

The following conventions are used to specify the context free grammar behind Konig:

Label	**Description**
`ID`	An identifier. It can be either a variable name or a function name
`EXP`	An expression, which returns a value
`STMT`	A statement, which does not return a value
`if, and, for ...`	Any lowercase word is understood to represent the corresponding reserved keyword

3.1.1 Identifiers

Identifiers in Konig are made up of any lowercase or uppercase ASCII letter, any decimal number, or the underscore character. However identifiers must start with a lowercase or uppercase ASCII letter.

```
// valid identifiers
int one_two = 0;
int two33 = 0;
int OneTwoTree = 0;

// invalid identifiers
int 1two = 0;
```

```
int _two = 0;
int !@# = 0;
```

Identifiers therefore match the regular expression `[a-zA-Z][a-zA-Z0-9_]*`

Grammar: `TYPE ID = EXP`

3.1.2 Comments

Comments are specified with a double forward-slash. Multiline comments use a forward slash paired with an asterisk as a starting token, and an asterisk paired with a slash as an end token.

```
graph<int> g = new graph{} // this is a comment

/* this is multiline comment
and it keeps going
and going */
```

Grammar: `TYPE ID = new TYPE{EXP, EXP, …}`

3.1.3 Separators

Konig uses semicolons to separate expressions, curly braces to separate blocks of expressions, and parentheses to isolate expressions that take precedence, and override the default preference order.

```
int x = 0; // here ';' is acting as a separator
x = x + 1;

if (x) { // here '{' signifies the start of a code block
      x = 2;
} // here '}' signifies the end of a code block

x = (x + 1) * (x + 2) // here '(' and ')' override precedence
```

Grammar: `{ STMT; STMT; STMT ... }`

3.1.4 Literals

3.1.4.1 Boolean Literals

The boolean literal is represented in Konig by the keywords `true` and `false`.

```
bool x = true;
```

3.1.4.2 Integer Literals

The integer literal is represented in Konig by an arbitrary length sequence of digits each between 0 and 9.

```
int x = 1234;
```

3.1.4.3 Float Literals

The floating point literal is represented in Konig as either pair of integers joined by a period, accompanied by an optional exponent indicated as following:

```
// these are all equivalent floating point literals
float x = 12.34;
float y = 1.234e+1;
float z = 123.4e-1;
float w = 1.234e1;
float j = 1234.e-2;
float k = .1234e+2;
```

Floating point literals therefore match the following regular expression:

```
[0-9]\.[0-9]*([eE][\+-]?[0-9]+)?
```

3.1.4.4 String Literals

A string literal in Konig is defined by an arbitrary sequence of characters contained within double quotes.

```
string x = "Hello World";
```

Grammar:

```
TYPE =
    int
  | bool
  | float
  | string
  | edge
  | list<TYPE>
  | node<TYPE>
  | graph<TYPE>
```

3.2 Data Types

The Konig programming language supports several primitive data types. Some of these data types can be found in any programming language, such as `int`, `bool` and `float`. Other data types are specific to graph theory, such as `node`, `edge` and `graph`. The language is statically and strongly typed, so the type of each variable is explicitly specified at the time of declaration. For those composite data types that contain another primitive, such as a `node`, the type of that primitive is specified between angle brackets after the container's type, in a Java-like fashion.

3.2.1 Primitives

Data Type	**Description**	**Example**
`int`	A 4 byte integer type	`int x = 3;`
`bool`	A 1 bit boolean type	`bool x = false;`
`string`	A series of 1 byte ASCII characters	`string x = "Hello world";`
`float`	An 8 byte floating-point type	`float x = 1.234;`
`void`	A reference to a null-like type	`float x = void;`

3.2.2 Lists

The list in Konig is identified by the keyword `list` followed by the type of its contents in angle brackets. All elements of a list must be the same type as declared at initialization.

```
list<int> x;

x = [1, 2, 3];
```

Lists can be accessed through the square bracket syntax:

```
int y = x[2]; // returns 3
```

Get list length:

```
x.length;
```

Grammar:

```
TYPE<IND> ID = [EXP, EXP, … ];
IND[EXP] = EXP;
```

3.2.2.1 List Functions

A number of functions are built into the standard library to manipulate lists. Each function modifies the list directly, and only returns a status value.

Function signature	**Description**
`ko int append(list<T> lst, T elem)`	Appends `elem` to the end of the list, and returns a new list
`ko int pop(list<T> lst, int index)`	Removes the element at the index provided

3.2.3 Nodes

A single node in Konig is identified by the keyword `node`. When initialized, a node is passed a data member. Nodes are initialized with the keyword `new`.

```
node<string> x;

x = new node{"hello world"};
```

A node's data member can be accessed by:

```
string y;
y = x.val; // returns "hello world"
```

3.2.4 Graphs

A graph in Konig is identified by the keyword `graph`. When initialized, a graph does not take any arguments.Graphs are initialized with the keyword `new`.

```
graph x;
x = new graph{};
```

3.2.5 Edges

Edges cannot be directly initialized by the user. However, an edge object will be returned by the operations that manipulate edges.

3.3 Operators

3.3.1 Arithmetic Operators

Konig implements a set of operators that are specific to graph theory, such as `@` to add a node to a graph, and `!` to remove a node from a graph. These operators make it easy to create & compose graphs, both directed and undirected. In addition, Konig implements all classic arithmetic operators, and a set of comparison operators.

Operator	Operands	Return type	Description
`a @ g`	`a` is a `node` `g` is a `graph`	`graph`	Adds the node a to the graph g

`a ! g`			Removes the node a from the graph g

`a + b`
`a - b`
`a / b`
`a * b`
	`a` is an `int, float`
	`b` is an `int, float`
		`int, float`	Performs the corresponding arithmetic operation (sum, difference, float division, multiplication, increment, decrement)

`a > b`
`a < b`
`a => b`
`a <= b`
`a == b`
	`a` is any type
	`b` is any type
	`a` and `b` have the same type
		`bool`	Performs the corresponding comparison operation, and returns a boolean value

`a and b`
`a or b`
`not a`
	`a` is a `bool`
	`b` is a `bool`
		`bool`	Performs the corresponding boolean operation between boolean values

3.4 Graph Semantics

3.4.1 Graphs

Initialize an empty graph and assign it to a variable `g1`:

```
graph<int> g1;
g1 = new graph{};
```

3.4.2 Nodes in Graph

Initialize a node `n0` with a value of `0`:

```
node<int> n0;
n0 = new node{0};
```

Add a node `n0` to the graph `g1`:

```
n0 @ g1;
```

Delete node `n0` from graph `g1`:

```
n0 ! g1;
```

Return list of nodes from graph `g1`:

```
g1.nodes;
```

Return list of edges from graph `g1`:

```
g1.edges;
```

Return list of all nodes that can be accessed from node `n1`:

```
neighbors(g, n1);
```

3.4.3 Edges in Graph

Create an undirected edge between nodes `n0` and `n1` in graph `g`, with weight value 0:

```
setEdge(g, n0, n1, 0.0);
```

Create a directed edge from node `n1` to `n2` in graph `g`, with weight value 5:

```
setDirEdge(g, n1, n2, 5.0);
```

Update edge weight from node `n1` to `n2`:

```
updateEdge(g, n1, n2, 15.0);
```

Delete edge from node `n1` to `n2`:

```
deleteEdge(g, n1, n2);
```

Access edge object between nodes `n1` and `n2` :

```
edge e;

e = getEdge(g, n1, n2);
```

Get edge type, returns `false` for undirected edges and `true` for directed edges:

```
e.directed;
getEdge(g, n1, n2).directed;
```

Get edge weight:

```
e.weight;
getEdge(g, n1, n2).weight;
```

3.5 Keywords

The following keywords are reserved in Konig:

```
ko                  else
bool                while
float               return
char                true
graph               false
node                and
edge                or
list                not
for                 int
if                  void
new
```

3.6 Control Flow

3.6.1 While Loop

The while statement has the form:

```
while (EXP) { STMT; STMT; ... }
```

The statements inside the code block are executed multiple times. After each execution of the code block, the expression in parenthesis is evaluated, and if returning `true` the statements inside the block are executed again.

3.6.2 For Loop

The for statement has the form

```
for ( EXP; EXP; EXP ) { STMT; STMT; ... }
```

The first expression specifies initialization for the loop. The second expression specifies a test, evaluated before each iteration, which terminates the loop once it evaluates to `false`. The third expression is evaluated at every iteration. It usually contains an increment.

3.6.3 Conditionals

There are two forms of conditional statements in Konig:

```
if ( EXP )  { STMT; STMT; ... }
if ( EXP ) {STMT; STMT; ... } else { STMT; STMT; ... }
```

In both forms, if the expression within parentheses evaluates to `true`, then the code block is executed. The second form of conditional also features a second code block, followed by the keyword else. This code block is executed if the expression evaluates to `false`.

3.7 Functions

Functions in Konig are defined with the reserved keyword `ko`. The function arguments are specified inside the parentheses and after the function name. The return type is specified after the closing parenthesis.

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

Grammar:

```
ko TYPE ID ( TYPE ID, TYPE ID, ... ) { STMT; STMT; ... }
```

3.8 Standard Library

The Konig programming language features a rich standard library for creating and manipulating graphs.

Function signature	**Description**
`ko edge setEdge(graph g, node a, node b, float weight) {}`	Constructs an undirected edge between two nodes with a weight to the edge. Users can define a default weight of 0. Returns an error if given nodes do not exist or are not in the same graph.
`ko edge setDirEdge(graph g, node a, node b, float weight) {}`	Constructs a directed edge from node a to node b with a weight to the edge. Users can define a default weight of 0. Returns an error if given nodes do not exist or are not in the same graph.
`ko edge getEdge(graph g, node a, node b)`	Returns the edge object between node a and node b
`ko edge updateEdge(graph g, node a, node b, float weight) {}`	Updates the weight of the edge between node a and node b
`ko edge deleteEdge (graph g, node a, node b) {}`	Deletes the edge between node a and b
`ko list<node> neighbors(graph g, node n) {}`	Returns a list of all neighbors that node n can reach: nodes that node n has a directed edge to or an undirected edge with.
`ko int viz(graph g, string path) {}`	Visualize the graph g and write the output to a PDF file (indicate name with path argument).

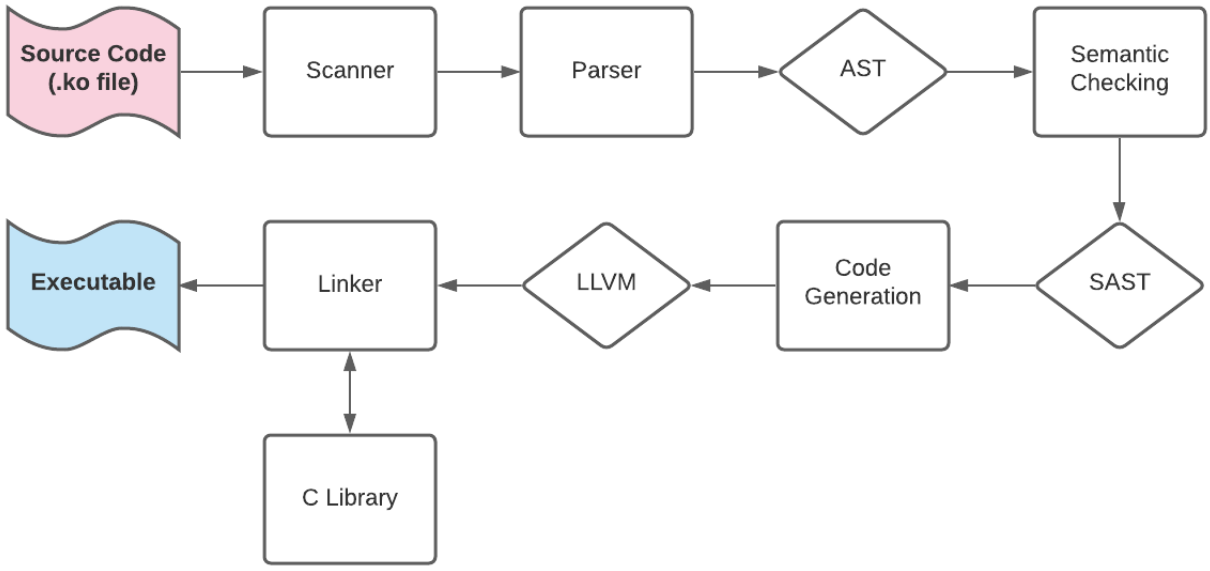
3.9 The `viz()` Function

The Konig programming language is connected to the Graphviz library, which allows the user to visualize any of the graphs created within Konig. The graphs are translated into the Graphviz

format by the `viz()` function, and then rendered as scalable vector graphics into a PDF file. We felt it was important for this feature to be present in the Konig language, because it allows the user to visually debug their program to a degree not possible before.

4. Architecture

4.1 Block Diagram



4.2 Scanner

scanner.mll

Our Ocamllex scanner takes in the source code and detects tokens for keywords, identifiers, operators, symbols, and values. Using these tokens, the scanner then passes them to the Parser in order to create an Abstract Syntax Tree (AST). During the lexical analysis, if the given source code contains unidentified keywords, an error will be thrown. A particularly tricky feature to implement in the scanner was single-line comments. Because the scanner by default strips all whitespace, a special detection case was developed for this purpose.

The scanner was implemented by Matteo.

4.3 Parser

parser.mly , ast.ml , prettyast.ml

The OCamlyacc parser reads the tokens passed in from the Scanner, and creates an AST representation of the source code. The goal is to make sure the program conforms to Konig's grammar, discard irrelevant information, and better understand the structure of the program. It was particularly interesting to implement Konig's composite types in the AST (i.e. `node<int>`), because they required a recursive definition of the grammar.

The parser was implemented by Matteo and Lord.

4.4 Semantic Checking

semant.mll , sast.ml , prettysast.ml

In the semantic checking stage, we check the AST generated from the Parser, assign a type to each expression, and pass a SAST (semantically checked abstract syntax tree) to the code generation step. We based our semantic checker on MicroC, and added a number of custom operators and functions. It was particularly challenging to figure out how to implement the semantic checking for the composite types, since some of the built-in functions do not know which flavor of composite type they will return until runtime. This problem was ultimately solved by allowing universal casting to and from void types.

Semantic checking was implemented by Matteo and Delilah.

4.5 Code Generation

codegen.ml

The code generation part of the compiler takes the SAST generated and returns an LLVM module. The LLVM module then gets compiled from the intermediate representation (IR) into native machine code. This component provided some interesting challenges by way of translating the Konig type system, which abstracts pointers completely and only deals with

values. Under the hood however, every piece of data in Konig is stored as a pointer to memory in the heap. Furthermore, since the more complicated logic is accomplished by C functions, and C does not support composite types the way Konig does natively, an extra amount of care was needed to make sure that each piece of data was cast to a void pointer before being passed to C, and then cast into the correct type of pointer before getting dereferenced.

Code Generation was implemented by Matteo and Delilah.

4.6 C Library and Linking

konig.c , konig.h , viz.c

As mentioned before, we implemented the data model for Konig and its more complex functions in C. Storage of each data type was also carefully constructed: graphs are stored as a list of edges and a list of nodes, where each edge holds a pointer to its endpoints. The lists themselves are implemented as doubly linked lists, where each node has a pointer to the next and previous one. The C implementation is also where Konig interfaces with Graphviz, the graph visualization library. However, extra care was taken to make sure that Konig can be successfully compiled without the library, for maximum compatibility.

This was implemented by Matteo and Lord.

5. Testing Plan

5.1 Testing Suite

Our team was most comfortable with Python as a scripting language. We built a small testing script in Python, stored in `/test.py`, which runs all of our tests automatically. The test cases are stored in the `/test` folder, and for each test case there is a pair of files: the `test-*.ko` file contains the Konig source code for the test case, while the corresponding `test-*.res` file contains the expected output.

The test suite starts by scanning the `/test` folder for any `*.ko` file. Then it compiles each source file, runs it, and compares it with the expected output using `diff`. If the compilation succeeds and there is no difference between actual and expected output, the test case is considered passed. If all tests pass, the suite lets the user know with a convenient message. If any test fails, the suite lets the user know which and how many tests failed.

The test suite is configured to be executed automatically after building the Konig compiler. In addition, the testing suite runs before any attempt is made to push code upstream to the repository, ensuring that all code that ends up in the main repository passes all test cases.

We chose the kind of test cases to implement based on each feature. Our aim was to have a few tests for each feature, and to not consider a feature completed until tests were written for it. To this end, we tried to always have at least barebones testing setup before merging any feature branch into the master branch.

5.2 Source to Target Examples

5.2.1 Example 1

This is the source program, in Konig:

```
ko int main() {
    node<string> a;
    node<string> b;
    node<string> c;
```

```
    graph<string> g;

    g = new graph{};
    a = new node{"A"};
    b = new node{"B"};
    c = new node{"C"};

    a @ g;
    b @ g;
    c @ g;

    setEdge(g, a, b, 1.0);
    setEdge(g, b, c, 1.0);
    setEdge(g, c, a, 1.0);

    viz(g, "sample.pdf");
}
```

This is the target result, in LLVM:

```
; ModuleID = 'Konig'
source_filename = "Konig"

%struct.Node = type { i8*, i8* }
%struct.Edge = type { i8*, %struct.Node*, %struct.Node*, i8, double }
%struct.Graph = type { i8*, %struct.Array*, %struct.Array* }
%struct.Array = type { i32, %struct.Elem*, %struct.Elem* }
%struct.Elem = type { i8*, %struct.Elem*, %struct.Elem* }

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@fmt.1 = private unnamed_addr constant [4 x i8] c"%g\0A\00", align 1
@fmt.2 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@str = private unnamed_addr constant [2 x i8] c"A\00", align 1
@str.3 = private unnamed_addr constant [2 x i8] c"B\00", align 1
@str.4 = private unnamed_addr constant [2 x i8] c"C\00", align 1
@str.5 = private unnamed_addr constant [11 x i8] c"sample.pdf\00", align 1

declare i32 @printf(i8*, ...)
declare i32 @print_node(%struct.Node*)
declare i32 @print_edge(%struct.Edge*)
declare i32 @print_graph(%struct.Graph*)
declare %struct.Array* @init_array()
declare i32 @append_array(%struct.Array*, i8*)
declare i8* @get_array(%struct.Array*, i32)
declare i32 @pop_array(%struct.Array*, i32)
declare %struct.Node* @init_node(i8*)
declare %struct.Graph* @add_node(%struct.Node*, %struct.Graph*)
declare %struct.Graph* @del_node(%struct.Node*, %struct.Graph*)
declare %struct.Edge* @set_edge(%struct.Graph*, %struct.Node*, %struct.Node*, double)
declare %struct.Edge* @set_dir_edge(%struct.Graph*, %struct.Node*, %struct.Node*, double)
declare %struct.Edge* @get_edge(%struct.Graph*, %struct.Node*, %struct.Node*)
declare %struct.Edge* @del_edge(%struct.Graph*, %struct.Node*, %struct.Node*)
declare %struct.Edge* @update_edge(%struct.Graph*, %struct.Node*, %struct.Node*, double)
declare %struct.Graph* @init_graph()
declare %struct.Array* @neighbors(%struct.Graph*, %struct.Node*)
declare i32 @visualize_graph(%struct.Graph*, i8*)
declare i8* @get_node_val(%struct.Node*)
declare i8* @get_node_id(%struct.Node*)
declare i1 @get_edge_directed(%struct.Edge*)
```

```
declare double @get_edge_weight(%struct.Edge*)
declare i8* @get_edge_id(%struct.Edge*)
declare i32 @get_array_length(%struct.Array*)
declare %struct.Array* @get_graph_nodes(%struct.Graph*)
declare %struct.Array* @get_graph_edges(%struct.Graph*)

define i32 @main() {
entry:
  %a = alloca %struct.Node*, align 8
  %b = alloca %struct.Node*, align 8
  %c = alloca %struct.Node*, align 8
  %g = alloca %struct.Graph*, align 8
  %init_node = call %struct.Graph* @init_graph()
  store %struct.Graph* %init_node, %struct.Graph** %g, align 8
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i32))
  %data = bitcast i8* %malloccall to i8**
  store i8* getelementptr inbounds ([2 x i8], [2 x i8]* @str, i32 0, i32 0), i8** %data, align 8
  %vdata = bitcast i8** %data to i8*
  %init_node1 = call %struct.Node* @init_node(i8* %vdata)
  store %struct.Node* %init_node1, %struct.Node** %a, align 8
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i32))
  %data3 = bitcast i8* %malloccall2 to i8**
  store i8* getelementptr inbounds ([2 x i8], [2 x i8]* @str.3, i32 0, i32 0), i8** %data3, align 8
  %vdata4 = bitcast i8** %data3 to i8*
  %init_node5 = call %struct.Node* @init_node(i8* %vdata4)
  store %struct.Node* %init_node5, %struct.Node** %b, align 8
  %malloccall6 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i32))
  %data7 = bitcast i8* %malloccall6 to i8**
  store i8* getelementptr inbounds ([2 x i8], [2 x i8]* @str.4, i32 0, i32 0), i8** %data7, align 8
  %vdata8 = bitcast i8** %data7 to i8*
  %init_node9 = call %struct.Node* @init_node(i8* %vdata8)
  store %struct.Node* %init_node9, %struct.Node** %c, align 8
  %a10 = load %struct.Node*, %struct.Node** %a, align 8
  %g11 = load %struct.Graph*, %struct.Graph** %g, align 8
  %add_node = call %struct.Graph* @add_node(%struct.Node* %a10, %struct.Graph* %g11)
  %b12 = load %struct.Node*, %struct.Node** %b, align 8
  %g13 = load %struct.Graph*, %struct.Graph** %g, align 8
  %add_node14 = call %struct.Graph* @add_node(%struct.Node* %b12, %struct.Graph* %g13)
  %c15 = load %struct.Node*, %struct.Node** %c, align 8
  %g16 = load %struct.Graph*, %struct.Graph** %g, align 8
  %add_node17 = call %struct.Graph* @add_node(%struct.Node* %c15, %struct.Graph* %g16)
  %g18 = load %struct.Graph*, %struct.Graph** %g, align 8
  %a19 = load %struct.Node*, %struct.Node** %a, align 8
  %b20 = load %struct.Node*, %struct.Node** %b, align 8
  %set_edge = call %struct.Edge* @set_edge(%struct.Graph* %g18, %struct.Node* %a19, %struct.Node* %b20,
double 1.000000e+00)
  %g21 = load %struct.Graph*, %struct.Graph** %g, align 8
  %b22 = load %struct.Node*, %struct.Node** %b, align 8
  %c23 = load %struct.Node*, %struct.Node** %c, align 8
  %set_edge24 = call %struct.Edge* @set_edge(%struct.Graph* %g21, %struct.Node* %b22, %struct.Node* %c23,
double 1.000000e+00)
  %g25 = load %struct.Graph*, %struct.Graph** %g, align 8
  %c26 = load %struct.Node*, %struct.Node** %c, align 8
  %a27 = load %struct.Node*, %struct.Node** %a, align 8
  %set_edge28 = call %struct.Edge* @set_edge(%struct.Graph* %g25, %struct.Node* %c26, %struct.Node* %a27,
double 1.000000e+00)
  %g29 = load %struct.Graph*, %struct.Graph** %g, align 8
  %visualize_graph = call i32 @visualize_graph(%struct.Graph* %g29, i8* getelementptr inbounds ([11 x i8],
[11 x i8]* @str.5, i32 0, i32 0))
  ret i32 0
}

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

5.2.1 Example 2

This is the source program, in Konig:

```
ko int main() {
    list<node<int>> l;
    node<int> x;
    node<int> y;
    node<int> z;
    int i;

    l = [
        new node{101},
        new node{202},
        new node{303}
    ];

    for (i=0; i < l.length; i = i+1) {
        print(l[i].val);
    }

    pop(l, 1);

    for (i=0; i < l.length; i = i+1) {
        print(l[i].val);
    }

}
```

This is the target result, in LLVM:

```
; ModuleID = 'Konig'
source_filename = "Konig"

%struct.Node = type { i8*, i8* }
%struct.Edge = type { i8*, %struct.Node*, %struct.Node*, i8, double }
%struct.Graph = type { i8*, %struct.Array*, %struct.Array* }
%struct.Array = type { i32, %struct.Elem*, %struct.Elem* }
%struct.Elem = type { i8*, %struct.Elem*, %struct.Elem* }

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@fmt.1 = private unnamed_addr constant [4 x i8] c"%g\0A\00", align 1
@fmt.2 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1

declare i32 @printf(i8*, ...)
declare i32 @print_node(%struct.Node*)
declare i32 @print_edge(%struct.Edge*)
declare i32 @print_graph(%struct.Graph*)
declare %struct.Array* @init_array()
declare i32 @append_array(%struct.Array*, i8*)
declare i8* @get_array(%struct.Array*, i32)
declare i32 @pop_array(%struct.Array*, i32)
declare %struct.Node* @init_node(i8*)
declare %struct.Graph* @add_node(%struct.Node*, %struct.Graph*)
declare %struct.Graph* @del_node(%struct.Node*, %struct.Graph*)
declare %struct.Edge* @set_edge(%struct.Graph*, %struct.Node*, %struct.Node*, double)
declare %struct.Edge* @set_dir_edge(%struct.Graph*, %struct.Node*, %struct.Node*, double)
declare %struct.Edge* @get_edge(%struct.Graph*, %struct.Node*, %struct.Node*)
declare %struct.Edge* @del_edge(%struct.Graph*, %struct.Node*, %struct.Node*)
```

```
declare %struct.Edge* @update_edge(%struct.Graph*, %struct.Node*, %struct.Node*, double)
declare %struct.Graph* @init_graph()
declare %struct.Array* @neighbors(%struct.Graph*, %struct.Node*)
declare i32 @visualize_graph(%struct.Graph*, i8*)
declare i8* @get_node_val(%struct.Node*)
declare i8* @get_node_id(%struct.Node*)
declare i1 @get_edge_directed(%struct.Edge*)
declare double @get_edge_weight(%struct.Edge*)
declare i8* @get_edge_id(%struct.Edge*)
declare i32 @get_array_length(%struct.Array*)
declare %struct.Array* @get_graph_nodes(%struct.Graph*)
declare %struct.Array* @get_graph_edges(%struct.Graph*)

define i32 @main() {
entry:
  %l = alloca %struct.Array*, align 8
  %x = alloca %struct.Node*, align 8
  %y = alloca %struct.Node*, align 8
  %z = alloca %struct.Node*, align 8
  %i = alloca i32, align 4
  %init_array = call %struct.Array* @init_array()
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %data = bitcast i8* %malloccall to i32*
  store i32 101, i32* %data, align 4
  %vdata = bitcast i32* %data to i8*
  %init_node = call %struct.Node* @init_node(i8* %vdata)
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %data2 = bitcast i8* %malloccall1 to i32*
  store i32 202, i32* %data2, align 4
  %vdata3 = bitcast i32* %data2 to i8*
  %init_node4 = call %struct.Node* @init_node(i8* %vdata3)
  %malloccall5 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32, i32* null, i32 1) to i32))
  %data6 = bitcast i8* %malloccall5 to i32*
  store i32 303, i32* %data6, align 4
  %vdata7 = bitcast i32* %data6 to i8*
  %init_node8 = call %struct.Node* @init_node(i8* %vdata7)
  %vdata9 = bitcast %struct.Node* %init_node to i8*
  %append_array = call i32 @append_array(%struct.Array* %init_array, i8* %vdata9)
  %vdata10 = bitcast %struct.Node* %init_node4 to i8*
  %append_array11 = call i32 @append_array(%struct.Array* %init_array, i8* %vdata10)
  %vdata12 = bitcast %struct.Node* %init_node8 to i8*
  %append_array13 = call i32 @append_array(%struct.Array* %init_array, i8* %vdata12)
  store %struct.Array* %init_array, %struct.Array** %l, align 8
  store i32 0, i32* %i, align 4
  br label %while

while:                                            ; preds = %while_body, %entry
  %i20 = load i32, i32* %i, align 4
  %l21 = load %struct.Array*, %struct.Array** %l, align 8
  %get_array_length = call i32 @get_array_length(%struct.Array* %l21)
  %tmp22 = icmp slt i32 %i20, %get_array_length
  br i1 %tmp22, label %while_body, label %merge

while_body:                                       ; preds = %while
  %l14 = load %struct.Array*, %struct.Array** %l, align 8
  %i15 = load i32, i32* %i, align 4
  %get_array = call i8* @get_array(%struct.Array* %l14, i32 %i15)
  %data16 = bitcast i8* %get_array to %struct.Node*
  %get_node_val = call i8* @get_node_val(%struct.Node* %data16)
  %data17 = bitcast i8* %get_node_val to i32*
  %data18 = load i32, i32* %data17, align 4
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32
0), i32 %data18)
  %i19 = load i32, i32* %i, align 4
  %tmp = add i32 %i19, 1
  store i32 %tmp, i32* %i, align 4
  br label %while

merge:                                            ; preds = %while
  %l23 = load %struct.Array*, %struct.Array** %l, align 8
  %pop_array = call i32 @pop_array(%struct.Array* %l23, i32 1)
```

```
  store i32 0, i32* %i, align 4
  br label %while24

while24:                                          ; preds = %while_body25, %merge
  %i36 = load i32, i32* %i, align 4
  %l37 = load %struct.Array*, %struct.Array** %l, align 8
  %get_array_length38 = call i32 @get_array_length(%struct.Array* %l37)
  %tmp39 = icmp slt i32 %i36, %get_array_length38
  br i1 %tmp39, label %while_body25, label %merge40

while_body25:                                     ; preds = %while24
  %l26 = load %struct.Array*, %struct.Array** %l, align 8
  %i27 = load i32, i32* %i, align 4
  %get_array28 = call i8* @get_array(%struct.Array* %l26, i32 %i27)
  %data29 = bitcast i8* %get_array28 to %struct.Node*
  %get_node_val30 = call i8* @get_node_val(%struct.Node* %data29)
  %data31 = bitcast i8* %get_node_val30 to i32*
  %data32 = load i32, i32* %data31, align 4
  %printf33 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32
0), i32 %data32)
  %i34 = load i32, i32* %i, align 4
  %tmp35 = add i32 %i34, 1
  store i32 %tmp35, i32* %i, align 4
  br label %while24

merge40:                                          ; preds = %while24
  ret i32 0
}

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

6. Project Plan

6.1 Planning Process

We met weekly on Thursdays as a team, to discuss the progression of our project and set goals for next week's meeting. In addition, depending on the goals we had set for the week, we would have working sessions to make progress on our project as a team, and make sure we were all on the same page. The tool we used to help our goal setting was Trello. This really helped keep us organized and on track to meet the deadlines we set for ourselves. We also had open communication via Facebook Messenger to ask each other questions and get help if we were stuck.

6.2 Development Flow

In the initial phases of developing our project, we met weekly to plan out the design and implementation of our language. We used MicroC as a foundation at the start of our project, to help guide us in developing the rest of the features of our language. We reached the code generation portion just as we were approaching the “Hello World” milestone. Following that, we started implementing our built-in functions and other key features of Konig. We stuck to our earlier plan of dividing up the tasks on Trello and implementing them one by one, which revealed itself as successful.

6.3 Development Tools

We used a `git` repository to synchronize and integrate our respective work, coupled with the powerful collaboration features offered by GitHub. Given that this semester was spent fully online, using a distributed version control system proved to be essential to a healthy development flow. We agreed that each new feature would require its own branch, in order to merge changes in an orderly fashion. Furthermore, a Pull Request was opened for each completed branch, and approval by another member of the team was required to merge it into the master branch.

6.4 Style Guide

We stuck to a few simple principles when putting together our style guide:

- Always use two spaces for indentation in OCaml, four in C and Konig
- Use `snake_case` to name variables
- Always leave a trailing new line character at the end of each source file

6.4 Roles & Responsibilities

Team Member	**Responsibility**
Matteo Sandrin	Scanner, Parser, Semantic Checking, Code Gen, C Library
Lord Crawford	Parser, C Library
Delilah Beverly	Semantic Checking, Code Gen

6.5 Timeline

January 21st	Ideating project ideas
February 3rd	Project proposal submitted
February 21st	Created our git repository and our first commit
February 24th	Submitted our LRM and Parser
March 24th	First successful codegen
April 25th	Project Presentation
April 26th	Final Report

6.5 Project Git Log

```
commit f949c8ad464a820a3ccf065b85eec474483bf793 (HEAD -> master, origin/master, origin/HEAD)
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Mon Apr 26 16:19:00 2021 -0400

    make graphviz optional

commit 00de718efb3c8437daa316905da1c5738257090f
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Mon Apr 26 15:17:04 2021 -0400

    update compile.sh to work with ubuntu

commit 46f0306318ef1f40a47ecb8441495c13966bac2d
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Mon Apr 26 15:08:59 2021 -0400

    remove free()

commit e7f062f3f61d21bae623919879f24d6f1c54ef79
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Mon Apr 26 15:06:14 2021 -0400

    remove forced string cast

commit 4ea8c68715362a14d15a54737e2e1fbfe1e92ff8
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Mon Apr 26 12:37:02 2021 -0400

    add make test recipe

commit bfe3a714a3052d71e36b60ca091bf25de7787f11
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Mon Apr 26 10:11:09 2021 -0400

    fix readme

commit 5075e00e5233ac7688806818e4cdcf1c8a6a4f9e
Merge: 309fe80 4cc78c7
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Mon Apr 26 10:08:54 2021 -0400

    Merge pull request #20 from matteosandrin/viz

    Fix readme

commit 4cc78c7bce98c2748bb4a94adeaeb5563386e5b3 (origin/viz)
Merge: 0fd2209 309fe80
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Mon Apr 26 10:06:57 2021 -0400

    Merge branch 'master' of https://github.com/matteosandrin/konig into viz

commit 0fd220919c39093dffcc4ce3862f9329521baf40
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Mon Apr 26 10:06:43 2021 -0400
```

```
    fix readme

commit 309fe80c895886af7ded145918adfe02a34309ce
Merge: 4405084 6deefab
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Mon Apr 26 09:47:26 2021 -0400

    Merge pull request #19 from matteosandrin/viz

    Graph visualization functionality

commit 6deefabb2de8c106f1c866512728c7451f8a83ee
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Mon Apr 26 09:46:52 2021 -0400

    update graphviz warning string

commit 4c60dcf9ef81102e2f706df92eea514a5957b185
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Mon Apr 26 09:36:47 2021 -0400

    update build script to make Graphviz optional

commit 328ab19585c50d4f3fd1563985969b9868e51622
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Sun Apr 25 13:27:04 2021 -0400

    implement graph visualization function

commit 4405084afd1227f20e614abf6ead453a6e0a500f
Merge: b5c1653 92db594
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Sun Apr 25 11:13:26 2021 -0400

    Merge pull request #17 from matteosandrin/graph-utility-functions

    Graph Utility Functions

commit 92db5940f32c1b907696f0d9850bf9631907516f (origin/graph-utility-functions)
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Sun Apr 25 11:13:02 2021 -0400

    implement g.edges

commit 6803f35a4fde2c0052708f27aea78349a62f2689
Merge: 492283e b5c1653
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Sun Apr 25 11:00:39 2021 -0400

    Merge branch 'master' into graph-utility-functions

commit b5c165337c12aa3a4bef16065eb6c0c3dce09bdb
Merge: 0231683 9ac1d76
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Sun Apr 25 11:00:16 2021 -0400

    Merge pull request #18 from matteosandrin/neighbors-function
```

```
    Neighbors function

commit 9ac1d76e03dbbf5b04acbf13228f80ddc9ce4287 (origin/neighbors-function)
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Sun Apr 25 10:56:05 2021 -0400

    implement neighbors function

commit 492283ed19b0ee0f06416556b64c3b69cd805434
Merge: 43cd208 0231683
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Sat Apr 24 21:52:35 2021 -0400

    Merge branch 'master' into graph-utility-functions

commit 02316831fed9b54ed88f6cf2bc7549545b533c66
Merge: eff3124 86b3cf8
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Sat Apr 24 21:52:16 2021 -0400

    Merge pull request #16 from matteosandrin/list-utility-functions

    List utility functions (append & pop)

commit 43cd208169d14147009ef1603fd133c168d38a0b
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Sat Apr 24 21:51:17 2021 -0400

    fix types

commit 86b3cf8d9aa9177a1d13fd32085b30b925e82c25 (origin/list-utility-functions)
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Sat Apr 24 21:39:06 2021 -0400

    implement array pop

commit 25f958cddf441d3eab7cdecb3976ab130ef490ae
Author: Lord Crawford <lrc2161@columbia.edu>
Date:   Sat Apr 24 21:29:36 2021 -0400

    .nodes feature for graphs

commit 6ca4f305bbebf81afcf01500b7f278fa94ac6c4c
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Sat Apr 24 21:25:48 2021 -0400

    implement array append() function

commit eff31243efdafa3773c3fa6338b563ab4747d409
Merge: 8d6a030 34cf134
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Sat Apr 24 20:48:45 2021 -0400

    Merge pull request #15 from matteosandrin/graph-utility-functions

    Convert Graph to composite Type
```

```
commit 34cf134e68a2c0f0a6b969f445d741a7e53cff7f
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Sat Apr 24 20:46:10 2021 -0400

    implement array length and printEdge

commit ba6ea7479a79447faf7e96537ad0450ce198e972
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Sat Apr 24 11:03:09 2021 -0400

    convert graph into a composite type (i.e. graph -> graph<int>)

commit 8d6a0301d96e9a282198f1f3b62343215223e079 (origin/list-features)
Merge: 36d3809 e39090b
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Sat Apr 24 10:25:58 2021 -0400

    Merge pull request #14 from matteosandrin/edge-library-functions

    Edge library functions

commit e39090b18cec97d7446ef8d96eff964fc986fd2f
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Sat Apr 24 10:24:29 2021 -0400

    implement updateEdge

commit 28448be594b7352cd55677b54642c05cbaae2c8d
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Sat Apr 24 10:14:45 2021 -0400

    add testing, change dot operator precendence

commit ebbaa7a3db8979a7183b2c46a6734feb0b5982f0
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Sat Apr 24 10:01:03 2021 -0400

    implement deleteEdge and expose edge.id and node.id

commit 503c2534999df56649c30e65692186fc421cd74d
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Fri Apr 23 17:56:20 2021 -0400

    implement getEdge() function

commit 36d38098ae32f1f511a70010e95f09085f5b98ce
Merge: c809977 f01cc60
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Fri Apr 23 16:52:26 2021 -0400

    Merge pull request #13 from matteosandrin/str-literal

    String Literal

commit f01cc60493e76f792a7579b40bec5359a55ba8fe (origin/str-literal)
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Fri Apr 23 16:50:31 2021 -0400
```

```
    add further testing

commit 6638501e5490e17a311de1cc30f3825cd51000bf
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Fri Apr 23 16:44:42 2021 -0400

    create string type

commit 909e0d1130969197921393be45f9bc6387f47484
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Fri Apr 23 16:27:21 2021 -0400

    implement printString() function

commit d1dc739ddcd75ff8a29d0edc358c24753b459843
Merge: dcf9826 c809977
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Fri Apr 23 16:14:49 2021 -0400

    Merge branch 'master' into str-literal

commit c8099777d1ef52a6a2e0070a3b3a9812b9dc3c0d
Merge: c7eb4dd fca49fe
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Fri Apr 23 15:58:42 2021 -0400

    Merge pull request #12 from matteosandrin/organize-dirs

    Organize Directory Structure

commit fca49fe76a558f155d4c63f250f317350549883b (origin/organize-dirs)
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Fri Apr 23 15:58:13 2021 -0400

    fix makefile

commit 30853f01835602ba09403c87fd0a8d30d9c9798c
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Fri Apr 23 15:55:51 2021 -0400

    move SAST and AST tests

commit fe95dba53d37c3b8120dc72789b389abdbc9e770
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Fri Apr 23 15:54:12 2021 -0400

    organize folder structure

commit c7eb4ddbe05ef74fd37c235129e5c853b0fd192c
Merge: 7d60a44 5980939
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Fri Apr 23 15:40:44 2021 -0400

    Merge pull request #11 from matteosandrin/db_dotnotation

    Dot Notation

commit 5980939bf3f699153a2115431970c4652afa171d (origin/db_dotnotation)
```

```
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Fri Apr 23 15:40:06 2021 -0400

    add further testing

commit ec88e09e4ebb8c66778f934682726d9e3f5e107c
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Fri Apr 23 15:23:34 2021 -0400

    add more tests

commit 39aaa6173575b2da551c2306977fd7466b84c8e4
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Fri Apr 23 15:12:45 2021 -0400

    change edge.type to edge.directed

commit 5d8e38aa1759e10daf7323587f48a2db14a79888
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Fri Apr 23 15:03:22 2021 -0400

    implement codegen component of dot notation

commit c3f5b89e9f6875dee235628597ac841b3f507698
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Fri Apr 23 14:12:31 2021 -0400

    rename variables

commit 19d201e070e6a5a4ff6705897d8202f1cf4c1b89
Merge: 037ff24 7d60a44
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Fri Apr 23 14:04:56 2021 -0400

    Merge branch 'master' into db_dotnotation

commit dcf98264baf58aa63d221070b1d340c6073fe3e0
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Fri Apr 23 12:34:25 2021 -0400

    add str type to codegen

commit 037ff24fd1f339d90c275c607b39e2f3834614c3 (db_dotnotation)
Author: delilah beverly <db3250@barnard.edu>
Date:   Fri Apr 23 12:28:00 2021 -0400

    semantly checked dot notation

commit 7d60a44377099581ebe979ff9c6962af4795d382
Merge: 69829c9 30301fe
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Fri Apr 23 10:53:45 2021 -0400

    Merge pull request #10 from matteosandrin/edge-literal

    Edge initializer, setEdge() function, setDirEdge() function

commit 30301fe85c9eb7d70e970b82598f2d2009b55bb7 (origin/edge-literal)
```

```
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Fri Apr 23 10:53:00 2021 -0400

    fix spacing

commit 4e90f8dd07d5a58ce56091170d8c0ac7c90eea1a
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Fri Apr 23 10:51:02 2021 -0400

    add tests

commit da0822642c66c51708d0154ac1508e15a305b4dd
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Fri Apr 23 10:43:01 2021 -0400

    hook setEdge and dirEdge into codegen

commit d62d14c0e26c686e43bdd4894c1c0ca718d2eba6
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Fri Apr 23 09:56:42 2021 -0400

    change function name to camel case

commit 3b6db83995ec5835bc9b9bc6869fd8b92e03f53b
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Fri Apr 23 09:51:56 2021 -0400

    setEdge and setDirEdge in c

commit fe8cc30420b0ce3ebda1a09a4b94371385d985d2
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Fri Apr 23 09:34:21 2021 -0400

    write edge init code

commit 69829c90988dacff5878d1b1a41a8652cd691c67
Merge: 6f1a395 6776ba9
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Fri Apr 23 00:15:15 2021 -0400

    Merge pull request #9 from matteosandrin/add-del-node

    Implement AddNode and DelNode operators

commit 6776ba90a9b2b4147b882bcf62c3746b24714693 (origin/add-del-node)
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Fri Apr 23 00:14:02 2021 -0400

    change argument name

commit f509492a658a84f3fb691381b277053d11cd3583
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Fri Apr 23 00:05:43 2021 -0400

    implement DelNode operator

commit b90d47d2d7bc4d6c7b612e7ec6e981316ebc3875
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
```

```
Date:   Thu Apr 22 23:33:21 2021 -0400

    implement AddNode and convert array wrapper to linked list

commit 1ad96fbcf33ff9a512ac6ba8373703dccf27f610
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Thu Apr 22 20:47:20 2021 -0400

    implement print_node() and print_graph() functions

commit c9e245634d24dbad7e7bbc637c354d50b26f5544
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Thu Apr 22 17:19:05 2021 -0400

    add random id to each node

commit 6f1a395993609f92f8cd2f0e8e39eb0672cf209c
Merge: 595d1ee 4cfb2f6
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Thu Apr 22 16:57:50 2021 -0400

    Merge pull request #8 from matteosandrin/graph-literal

    Graph literal

commit 4cfb2f64c98fb413cbfbbeff4e5b2b621af12b4e (origin/graph-literal)
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Thu Apr 22 16:50:59 2021 -0400

    implement graph literal

commit 595d1eefef3a9d9d508219d58272f7f1eaf1dc76
Merge: 1338e31 8fe06ef
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Thu Apr 22 16:28:03 2021 -0400

    Merge pull request #7 from matteosandrin/node-literal

    Node literal

commit 8fe06efaff0255cf75321842cd38278d2ec329cb (origin/node-literal)
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Thu Apr 22 16:21:58 2021 -0400

    add node literal test

commit 91c88b38a16a875e161f5eb0d20b60a696ca5e38
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Thu Apr 22 16:14:08 2021 -0400

    implement node literal initialization

commit 1338e316c38dcfbd9d2bb35cf7952356d2e7433c
Merge: 64a1a06 08ccbb2
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Thu Apr 22 15:30:40 2021 -0400

    Merge pull request #6 from matteosandrin/sast-lists
```

```
    Implement list literals, list indexing and a basic test suite

commit 08ccbb2e8a134357a0d9485b9a54b80db72269b5 (origin/sast-lists)
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Thu Apr 22 15:25:42 2021 -0400

    write basic testing suite

commit be21b77639089a5140b5fd93faae40decd46189b
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Thu Apr 22 14:41:39 2021 -0400

    re-write compile script

commit e605349caac63ceb8fda407e9b8c02aaf47c2533
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Thu Apr 22 13:16:15 2021 -0400

    implement array access through indexing

commit e0eaf231784e4bc1e9f57bd1dbb1e3e96eef4814
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Thu Apr 22 11:23:42 2021 -0400

    implement array initialization

commit f1e5efdbb0347e8b9e350ea0f3f219d1d59b56f2
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Thu Apr 22 09:35:55 2021 -0400

    linked C code

commit 5c4cb5198bc4ed1ead487e34a872df46049439e9
Author: delilah beverly <db3250@barnard.edu>
Date:   Wed Apr 21 19:49:25 2021 -0400

    updated correctly

commit 7d9e6cbc0fffe3091fbd72b3804a8c6071928485
Author: delilah beverly <db3250@barnard.edu>
Date:   Wed Apr 21 18:57:34 2021 -0400

    dot notation parser

commit 1fec3dfe1d83505c90ed7c8d88db087a3af68d5f
Author: delilah beverly <db3250@barnard.edu>
Date:   Wed Apr 21 18:55:46 2021 -0400

    dot notation parser

commit cbffdc1353a9e2bf8578c51ae5ef27e77098aebb
Author: delilah beverly <db3250@barnard.edu>
Date:   Tue Apr 6 21:42:10 2021 -0400

    dot notation parser

commit ee207e5810029d4a8ffb97309659b4cafb14f7de
```

```
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Mon Apr 5 15:34:00 2021 -0400

    Change makefile to allow custom output path

commit a279795ffd35f36345d65d51f825867f23f4f709
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Mon Apr 5 15:26:32 2021 -0400

    Add more informative error

commit 5fdb0c3fe8ab770d14140fa6cd21a66743e0212c
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Mon Apr 5 15:20:31 2021 -0400

    Implement semantic checking of list literals

commit 64a1a0676fe6caad9edf05bb7a06bc88098aca29
Merge: c7282f7 8315a2e
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Wed Mar 24 21:24:16 2021 -0400

    Merge pull request #5 from matteosandrin/codegen-hello-world

    Implement "Hello World" code generation

commit 8315a2eb82eaf4a6daedf673cdae88748b08bfd5 (origin/codegen-hello-world)
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Wed Mar 24 21:23:46 2021 -0400

    Add readme

commit f1ca4306dbe8e7d66603715aec96b3b68d96d739
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Wed Mar 24 21:18:09 2021 -0400

    Implemented basic code generation

commit c7282f736730a2b939bc1717f15af2db4f6291b1
Merge: b5f49ea 273ad08
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Wed Mar 24 20:21:39 2021 -0400

    Merge pull request #4 from matteosandrin/semantic-checking

    Implement Semantic Checking & SAST

commit 273ad084047b324d7bcd07611f3a7fe693d8f968 (origin/semantic-checking)
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Wed Mar 24 16:00:19 2021 -0400

    Integrate semantic checking with konig.ml

commit 2625f3de347ae509281a54ab7be158e88bfdd994
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Wed Mar 24 15:04:56 2021 -0400

    Create semant.ml
```

```
commit 479f5461399eac52190bded7c1eb2ec4dbd06f22
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Wed Mar 24 14:59:05 2021 -0400

    Create sast.ml

commit b5f49ea91325d50d2287a1495bda709d2deaa04d
Merge: df03627 7394132
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Tue Mar 23 16:01:34 2021 -0400

    Merge pull request #3 from matteosandrin/ast

    Build the Abstract Syntax Tree

commit 73941328edb686dd84c968f3ddeded247437e27a (origin/ast)
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Tue Mar 23 10:54:15 2021 -0400

    Separate VarargLit into different components

commit 3b11e252d7ae8ee3222e679098dbc62c37e6d1ba
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Tue Mar 23 10:31:21 2021 -0400

    Connect Parser to AST and implement pretty printing

commit 9694b7a7723565250196afc83ea89a83adfc09d1
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Tue Mar 23 09:41:51 2021 -0400

    Add pretty printing of AST and the konig.ml file

commit dc799be6ff3ef27c24024252baf11c4d79ceb7d9
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Tue Mar 23 09:09:04 2021 -0400

    Add expressions to AST

commit 7c15f60e131b4bb236ebc0c63a67d8aac5140212
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Tue Mar 23 08:55:26 2021 -0400

    Add operations and types to AST

commit 4e5d677b9781e79239e2b24c3a5dde4184b39d84
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Tue Mar 23 08:44:08 2021 -0400

    Create ast.ml file

commit df036278f236c009f029f4a243e1109d268295cc
Merge: 1137620 70314c8
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Wed Feb 24 23:23:24 2021 -0500

    Merge pull request #2 from matteosandrin/additional-parser-features
```

```
    Additional parser features

commit 70314c8b698dbed401aa5a344a7eb56d239ea1d7 (origin/additional-parser-features)
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Wed Feb 24 22:03:57 2021 -0500

    Add edge primitive

commit e1148fd0b4dca16d8e7bb4e9cb4dbef579bfdb72
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Wed Feb 24 21:54:01 2021 -0500

    implement member access and assignment with dot ('.')

commit 113762094f8a5cc84f233e792d4338ce741dc006
Merge: 389ef00 80c1959
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Wed Feb 24 20:31:44 2021 -0500

    Merge pull request #1 from matteosandrin/parser

    Implement Parser

commit 80c195913d3fb21eeb2613cad864f58522e20eb1 (origin/parser)
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Wed Feb 24 20:22:00 2021 -0500

    implement initialization of graphs and nodes

commit e2190c873198513971685b001662fd2d6a654ce1
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Wed Feb 24 15:21:29 2021 -0500

    Implement node & graph types

commit c0779db38772041044294fc620d412eae06af51c
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Wed Feb 24 15:04:06 2021 -0500

    Implement list literal

commit fea74b7fa4c5c4426a925a81a41a877f6a070174
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Wed Feb 24 14:49:08 2021 -0500

    Add grammar for list indexing (ex: l[0])

commit 1099fb0a0a77174c081c86877f7ef1015205986d
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Wed Feb 24 14:42:09 2021 -0500

    Implement char type

commit 55023b5f0903188c08cb4ea54bcf4d3871fc39bd
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Wed Feb 24 14:39:18 2021 -0500
```

```
    implement list type

commit 20d7f139689c226c995e2899ed4da07dc8c20597
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Wed Feb 24 14:14:31 2021 -0500

    set function keyword to 'ko'

commit 06eea9d59dc4a5cc508da71832ffcf1b437ae272
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Wed Feb 24 14:09:07 2021 -0500

    implement string literal

commit 05a6f82d8e0e9b5b33f2921599f51b56f9e1c41f
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Mon Feb 22 17:35:07 2021 -0500

    Implement single-line comments

commit 19182d91a7512f1422256b08a5177a256e98e28e
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Mon Feb 22 15:20:48 2021 -0500

    Add operators '@' and '!' to add and remove a node respectively

commit 1c0c200f6035e89cb25636727e4ca06977e06170
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Mon Feb 22 15:12:03 2021 -0500

    Change logical operators (&& -> 'and')

commit 271150b69eada9f4ac132f1ca62028fa5009f2b8
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Mon Feb 22 15:10:45 2021 -0500

    Add Makefile

commit 389ef00e9299d17a59ccc786049b90f72ecbf6db
Author: Matteo Sandrin <matteo.sandrin@hotmail.it>
Date:   Mon Feb 22 14:29:35 2021 -0500
```

7. Lessons Learned

7.1 Matteo Sandrin

I think the single most important lesson that I learned during this project is how to prioritize effectively to reach a complex goal. There were lots of occasions on which I was tempted to track down the rabbit hole of some odd memory bug, but this project has forced me to accept that sometimes solving those kinds of problems has dramatically diminishing returns with respect to time invested. I instead learned to stay on the main path and get the critical stuff done first, leaving the quirky bugs for last, if there's time. Although I like to think that this project has taught me some good time management skills, I still think I could have done much of the work much earlier than I did. My message to future generations of PLT teams is the following: start writing code much earlier than you think is necessary, because it'll catch up with you incredibly fast!

7.2 Delilah Beverly

During this semester-long project I've learned more about my learning style and process. OCaml and the content of the course was very scary to me at first, and just the idea of building a whole compiler seemed very daunting in the beginning, but what really helped me to manage was to make sure to break down concepts and problems into smaller more digestible pieces. During the process of creating our compiler breaking down tasks into smaller subtasks made it a lot easier to learn and accomplish more. As opposed to being overwhelmed and stuck and feeling like you're not making progress.

7.3 Lord Crawford

Outside of this being my first encounter with a functional programming language and learning the deep intricacies of OCaml, something I learned was how important it is to establish an effective development flow amongst your team. While our team was only three members, it helped immensely to split changes amongst ‘feature branches’ on Github and do regular meetings updating each other of our progress, things we were confused about, and generally staying on the same page with one another. We also utilized Trello as our project management tool, I highly

recommend for future teams to use something of this nature (or the ‘Projects’ tab on Github) to manage everyone tasks and stay aligned in your goals. In addition, I recommend future PLT students to take extensive time to learn OCaml fundamentals at the beginning of this project and spend sessions learning it together as it will help clear confusion early on.

Appendix

Scanner

scanner.mll

```
(* Ocamllex scanner for Konig *)

{ open Parser }

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

rule token = parse
 [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"     { multi_comment lexbuf }      (* Multiline comments *)
| "//"     { comment lexbuf }           (* Single line comments *)
| '('      { LPAREN }
| ')'      { RPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
| '['      { LSQUARE }
| ']'      { RSQUARE }
| ';'      { SEMI }
| ','      { COMMA }
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '='      { ASSIGN }
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
| ">"      { GT }
| ">="     { GEQ }
| "and"    { AND }
| "or"     { OR }
| "not"    { NOT }
```

```
| "if"     { IF }
| "else"   { ELSE }
| "for"    { FOR }
| "while"  { WHILE }
| "ko"     { KO }
| "new"    { NEW }
| "return" { RETURN }
| "int"    { INT }
| "bool"   { BOOL }
| "float"  { FLOAT }
| "string" { STRING }
| "list"   { LIST }
| "node"   { NODE }
| "edge"   { EDGE }
| "graph"  { GRAPH }
| "void"   { VOID }
| "true"   { BLIT(true)  }
| "false"  { BLIT(false) }
| "@"      { ADDNODE }
| "!"      { DELNODE }
| digits as lxm { LITERAL(int_of_string lxm) }
| digits '.' digit* ( ['e' 'E'] ['+' '-']? digits )? as lxm { FLIT(lxm) }
| "."     { DOT }
| '"' ([^'"']* as lxm) '"' { STRLIT(lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*     as lxm { ID(lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

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

and comment = parse
 ['\r' '\n'] { token lexbuf }
| _          { comment lexbuf }
```

Parser

parser.mly

```
/* Ocamlyacc parser for Konig */

%{
open Ast
%}

%token SEMI LPAREN RPAREN LBRACE RBRACE LSQUARE RSQUARE COMMA DOT PLUS MINUS TIMES
DIVIDE ASSIGN
%token ADDNODE DELNODE
%token NOT EQ NEQ LT LEQ GT GEQ AND OR
%token KO NEW RETURN IF ELSE FOR WHILE INT BOOL FLOAT STRING LIST NODE EDGE GRAPH
VOID
%token <int> LITERAL
%token <bool> BLIT
%token <string> ID FLIT
%token <string> STRLIT
%token EOF

%start program
%type <Ast.program> program

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

%%

program:
```

```
 decls EOF { $1 }

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

fdecl:
  KO typ ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list RBRACE
    { { typ = $2;
  fname = $3;
  formals = List.rev $5;
  locals = List.rev $8;
  body = List.rev $9 } }

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

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

typ:
   INT    { Int   }
 | BOOL   { Bool  }
 | FLOAT  { Float }
 | STRING { Str }
 | EDGE   { Edge }
 | VOID   { Void  }
 | LIST LT typ GT  { List($3)  }
 | NODE LT typ GT  { Node($3)  }
 | GRAPH LT typ GT { Graph($3) }

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

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

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

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

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

expr:
   LITERAL          { Literal($1)            }
 | FLIT             { Fliteral($1)           }
 | BLIT             { BoolLit($1)            }
 | STRLIT           { StrLit($1)             }
 | ID               { Id($1)                 }
 | expr PLUS   expr { Binop($1, Add,   $3)   }
 | expr MINUS  expr { Binop($1, Sub,   $3)   }
 | expr TIMES  expr { Binop($1, Mult,  $3)   }
 | expr DIVIDE expr { Binop($1, Div,   $3)   }
 | expr 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 DOT    ID   { Prop($1, $3)           } // (Dot notation)
 | expr ADDNODE expr { Binop($1, Addnode, $3) } // (add node)
 | expr DELNODE expr { Binop($1, Delnode, $3) } // (delete node)
 | MINUS expr %prec NOT { Unop(Neg, $2)      }
 | NOT expr         { Unop(Not, $2)          }
```

```
 | ID ASSIGN expr   { Assign($1, $3)         }
 | ID LPAREN args_opt RPAREN { Call($1, $3)  }
 | LPAREN expr RPAREN { $2                   }
 | ID LSQUARE expr RSQUARE { Index($1, $3) } // (list indexing)
 | LSQUARE args_opt RSQUARE { ListLit($2) } // (list literal)
 | NEW NODE LBRACE args_opt RBRACE { NodeLit($4) } // (node literal)
 | NEW GRAPH LBRACE args_opt RBRACE { GraphLit($4)  } // (graph literal)

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

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

AST

ast.ml

```
(* Abstract Syntax Tree *)

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

type uop = Neg | Not

type typ = Int | Bool | Float | Void |
           Edge | Str |
           List of typ |
           Node of typ |
           Graph of typ

type bind = typ * string

type expr =
    Literal of int
  | Fliteral of string
  | BoolLit of bool
  | StrLit of string
  | ListLit of expr list
  | NodeLit of expr list
  | GraphLit of expr list
  | Id of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Assign of string * expr
  | Call of string * expr list
  | Index of string * expr (* access an item in an array *)
  | Prop of expr * string (* access an edge or nodes data memeber *)
  | Noexpr

type stmt =
    Block of stmt list
  | Expr of expr
  | Return of expr
```

```
 | If of expr * stmt * stmt
 | For of expr * expr * expr * stmt
 | While of expr * stmt

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

type program = bind list * func_decl list
```

prettyast.ml

```
(* Pretty-printing functions *)
open Ast

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

let string_of_uop = function
   Neg -> "-"
 | Not -> "not"
```

```
let rec string_of_expr = function
   Literal(l) -> string_of_int l
 | Fliteral(l) -> l
 | BoolLit(true) -> "true"
 | BoolLit(false) -> "false"
 | StrLit(l) -> "\"" ^ l ^ "\""
 | ListLit(el) ->
   "[" ^ String.concat ", " (List.map string_of_expr el) ^ "]"
 | NodeLit(el) ->
   "new node{" ^ String.concat ", " (List.map string_of_expr el) ^ "}"
 | GraphLit(el) ->
   "new graph{" ^ String.concat ", " (List.map string_of_expr el) ^ "}"
 | Id(s) -> s
 | Binop(e1, o, e2) ->
     string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
 | Unop(o, e) -> string_of_uop o ^ string_of_expr e
 | Assign(v, e) -> v ^ " = " ^ string_of_expr e
 | Call(f, el) ->
     f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
 | Index(v, e) ->
     v ^ "[" ^ string_of_expr e ^ "]"
 | Prop(e, s) ->
     string_of_expr e ^ "." ^ s
 | Noexpr -> ""

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

let rec string_of_typ = function
   Int -> "int"
 | Bool -> "bool"
```

```
 | Float -> "float"
 | Void -> "void"
 | Str -> "string"
 | Edge -> "edge"
 | Graph(t) -> "graph<" ^ string_of_typ t ^ ">"
 | List(t) -> "list<" ^ string_of_typ t ^ ">"
 | Node(t) -> "node<" ^ string_of_typ t ^ ">"

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

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

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

pretty_ast_test.ko

```
ko int main() {
   int x;
   list<int> y;
   list<node<bool>> z; // complex type
   node<int> w;
   x = 1; // int literal
   z = [1, 2, 3]; // list literal
   w = new node{1}; // node literal
   z = z[3]; // list indexing
   z = x @ y; // add node
   z = x ! y; // remove node
   z = not (a and b or c); // boolean operators
   z = call(a, b, c); // function call
   z = "hello world"; // string literal
```

```
}
```

SAST

sast.ml

```
(* Abstract Syntax Tree *)
open Ast

type sexpr = typ * sx
and sx =
   SLiteral of int
 | SFliteral of string
 | SBoolLit of bool
 | SStrLit of string
 | SListLit of sexpr list
 | SNodeLit of sexpr list
 | SGraphLit of sexpr list
 | SId of string
 | SBinop of sexpr * op * sexpr
 | SUnop of uop * sexpr
 | SAssign of string * sexpr
 | SCall of string * sexpr list
 | SIndex of string * sexpr (* access an item in an array *)
 | SProp of sexpr * string (* access an edge or nodes data memeber *)
 | SNoexpr

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

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

```
type sprogram = bind list * sfunc_decl list
```

prettysast.ml

```
(* Pretty-printing functions *)

open Sast
open Prettyast

let rec string_of_sexpr (t, e) =
 "(" ^ string_of_typ t ^ " : " ^ (match e with
   SLiteral(l) -> string_of_int l
 | SFliteral(l) -> l
 | SBoolLit(true) -> "true"
 | SBoolLit(false) -> "false"
 | SStrLit(l) -> "\"" ^ l ^ "\""
 | SListLit(el) ->
   "[" ^ String.concat ", " (List.map string_of_sexpr el) ^ "]"
 | SNodeLit(el) ->
   "new node{" ^ String.concat ", " (List.map string_of_sexpr el) ^ "}"
 | SGraphLit(el) ->
   "new graph{" ^ String.concat ", " (List.map string_of_sexpr el) ^ "}"
 | SId(s) -> s
 | SBinop(e1, o, e2) ->
     string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_sexpr e2
 | SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
 | SAssign(v, e) -> v ^ " = " ^ string_of_sexpr e
 | SCall(f, el) ->
     f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^ ")"
 | SIndex(v, e) ->
   v ^ "[" ^ string_of_sexpr e ^ "]"
 | SProp(e, s) ->
   string_of_sexpr e ^ "." ^ s
 | SNoexpr -> ""
       ) ^ ")"

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

```
 | SExpr(expr) -> string_of_sexpr expr ^ ";\n";
 | SReturn(expr) -> "return " ^ string_of_sexpr expr ^ ";\n";
 | SIf(e, s, SBlock([])) ->
     "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s
 | SIf(e, s1, s2) ->  "if (" ^ string_of_sexpr e ^ ")\n" ^
     string_of_sstmt s1 ^ "else\n" ^ string_of_sstmt s2
 | SFor(e1, e2, e3, s) ->
     "for (" ^ string_of_sexpr e1  ^ " ; " ^ string_of_sexpr e2 ^ " ; " ^
     string_of_sexpr e3  ^ ") " ^ string_of_sstmt s
 | SWhile(e, s) -> "while (" ^ string_of_sexpr e ^ ") " ^ string_of_sstmt s

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

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

pretty_sast_test.ko

```
ko void main() {
   int x;
   x = 1;
   print(x);
}
```

semant.ml

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

```
open Ast
open Sast
open Prettyast
open Prettysast

module StringMap = Map.Make(String)

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

  Check each global variable, then check each function *)

let check (globals, functions) =

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

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

 check_binds "global" globals;

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

 (* Collect function declarations for built-in functions: no bodies *)
 let built_in_decls =
   let add_bind map (name, ty) = StringMap.add name {
     typ = Void;
     fname = name;
     formals = [(ty, "x")];
     locals = []; body = [] } map
   in List.fold_left add_bind StringMap.empty [
```

```
    ("print", Int);
    ("printb", Bool);
    ("printf", Float);
    ("printString", Str);
    ("printNode", Node(Void));
    ("printEdge", Edge);
    ("printGraph", Graph(Void));
  ]
in
let built_in_decls = StringMap.add "setEdge" {
  typ = Edge;
  fname = "setEdge";
  formals = [(Graph(Void), "g"); (Node(Void), "from"); (Node(Void), "to"); (Float,
"weight")];
  locals = [];
  body = [];
} built_in_decls
in
let built_in_decls = StringMap.add "setDirEdge" {
  typ = Edge;
  fname = "setDirEdge";
  formals = [(Graph(Void), "g"); (Node(Void), "from"); (Node(Void), "to"); (Float,
"weight")];
  locals = [];
  body = [];
} built_in_decls
in
let built_in_decls = StringMap.add "getEdge" {
  typ = Edge;
  fname = "getEdge";
  formals = [(Graph(Void), "g"); (Node(Void), "from"); (Node(Void), "to")];
  locals = [];
  body = [];
} built_in_decls
in
let built_in_decls = StringMap.add "deleteEdge" {
  typ = Edge;
  fname = "deleteEdge";
  formals = [(Graph(Void), "g"); (Node(Void), "from"); (Node(Void), "to")];
  locals = [];
  body = [];
```

```
 } built_in_decls
 in
 let built_in_decls = StringMap.add "updateEdge" {
   typ = Edge;
   fname = "updateEdge";
   formals = [(Graph(Void), "g"); (Node(Void), "from"); (Node(Void), "to"); (Float,
"weight")];
   locals = [];
   body = [];
 } built_in_decls
 in
 let built_in_decls = StringMap.add "append" {
   typ = Int;
   fname = "append";
   formals = [(List(Void), "array"); (Void, "data")];
   locals = [];
   body = [];
 } built_in_decls
 in
 let built_in_decls = StringMap.add "pop" {
   typ = Int;
   fname = "pop";
   formals = [(List(Void), "array"); (Int, "index")];
   locals = [];
   body = [];
 } built_in_decls
 in
 let built_in_decls = StringMap.add "neighbors" {
   typ = List(Node(Void));
   fname = "neighbors";
   formals = [(Graph(Void), "g"); (Node(Void), "n")];
   locals = [];
   body = [];
 } built_in_decls
 in
 let built_in_decls = StringMap.add "viz" {
   typ = Int;
   fname = "viz";
   formals = [(Graph(Void), "g"); (Str, "path")];
   locals = [];
   body = [];
```

```
} built_in_decls
in

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

(* Collect all function names into one symbol table *)
let function_decls = List.fold_left add_func built_in_decls functions
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 =
    match (lvaluet, rvaluet) with
      (* this allows us to cast a Node<int> to Node<void>, to support print_node *)
        (_, Void) -> lvaluet
      | (Void, _) -> rvaluet
      | (Node(_), Node(Void)) -> lvaluet
      | (Node(Void), Node(_)) -> rvaluet
      | (Graph(_), Graph(Void)) -> lvaluet
```

```
      | (Graph(Void), Graph(_)) -> rvaluet
      | (List(_), List(Void)) -> lvaluet
      | (List(Void), List(_)) -> rvaluet
      | (List(Node(_)), List(Node(Void))) -> lvaluet
      | (List(Node(Void)), List(Node(_))) -> rvaluet
      | _ -> 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

  (* check that all the expressions in the list have the same type as the first one
*)
  let check_all_types_same sexps err = match sexps with
      (typ, _) :: tail ->
        List.iter (fun (t, e) ->
          if t <> typ
          then raise (Failure (err)))
        tail; typ
    | [] -> Void
  in

  (* Return a semantically-checked expression, i.e., with a type *)
  let rec expr = function
      Literal  l -> (Int, SLiteral l)
    | Fliteral l -> (Float, SFliteral l)
    | BoolLit l  -> (Bool, SBoolLit l)
    | StrLit l   -> (Str, SStrLit l)
    | Noexpr     -> (Void, SNoexpr)
    | Id s       -> (type_of_identifier s, SId s)
    | ListLit(exps) as ex ->
```

```
      let sexps = (List.map (fun e -> expr e) exps) in
      let err = "expressions must all have the same type in " ^ string_of_expr ex
in
      (List(check_all_types_same sexps err), SListLit(sexps))
    | NodeLit(exps) ->
      let sexps = match (List.length exps) with
          1 -> (List.map (fun e -> expr e) exps)
        | _ -> raise ( Failure ("illegal number of arguments for new node{val}.
Must have exactly one argument"))
      in
      let typ = (fst (List.hd sexps))
      in (Node(typ), SNodeLit(sexps))
    | GraphLit(exps) ->
      let sexps = match (List.length exps) with
          0 -> (List.map (fun e -> expr e) exps)
        | _ -> raise ( Failure ("illegal number of arguments for new graph{}. Must
have exactly zero arguments"))
      in (Graph(Void), SGraphLit(sexps))
    | Assign(var, e) as ex ->
        let lt = type_of_identifier var
        and (rt, e') = expr e in
        let err = "illegal assignment " ^ string_of_typ lt ^ " = " ^
          string_of_typ rt ^ " in " ^ string_of_expr ex
        in (check_assign lt rt err, SAssign(var, (rt, e')))
    | Prop(e, prop) ->
      let (etyp, _) as e' = expr e in
      let pt = match (etyp, prop) with
          (Node t, "val") -> t
        | (Node t, "id") -> Str
        | (Edge, "directed") -> Bool
        | (Edge, "weight") -> Float
        | (Edge, "id") -> Str
        | (List t, "length") -> Int
        | (Graph t, "nodes") -> List(Node(t))
        | (Graph t, "edges") -> List(Edge)
        | (_, _) -> raise (Failure ("illegal property access"))
      in
      (pt, SProp (e', prop))
    | Unop(op, e) as ex ->
        let (t, e') = expr e in
        let ty = match op with
```

```
          Neg when t = Int || t = Float -> t
        | Not when t = Bool -> Bool
        | _ -> raise (Failure ("illegal unary operator " ^
                               string_of_uop op ^ string_of_typ t ^
                               " in " ^ string_of_expr ex))
        in (ty, SUnop(op, (t, e')))
    | Binop(e1, op, e2) as e ->
        let (t1, e1') = expr e1
        and (t2, e2') = expr e2 in
        (* All binary operators require operands of the same type *)
        let same = t1 = t2 in
        (* Determine expression type based on operator and operand types *)
        let ty = match op with
          Add | Sub | Mult | Div when same && t1 = Int   -> Int
        | Add | Sub | Mult | Div when same && t1 = Float -> Float
        | Equal | Neq            when same               -> Bool
        | Less | Leq | Greater | Geq
                   when same && (t1 = Int || t1 = Float) -> Bool
        | And | Or when same && t1 = Bool -> Bool
        | Addnode | Delnode when
          match (t1, t2) with
              (Node(subt1), Graph(subt2)) -> subt1 = subt1
            | _       -> false
          -> t1
        | _ -> raise (
      Failure ("illegal binary operator " ^
                    string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
                    string_of_typ t2 ^ " in " ^ string_of_expr e))
        in (ty, SBinop((t1, e1'), op, (t2, e2')))
    | Call(fname, args) as call ->
        let fd = find_func fname in
        let param_length = List.length fd.formals in
        if List.length args != param_length then
          raise (Failure ("expecting " ^ string_of_int param_length ^
                          " arguments in " ^ string_of_expr call))
        else let check_call (ft, _) e =
          let (et, e') = expr e in
          let err = "illegal argument found " ^ string_of_typ et ^
            " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr e
          in (check_assign ft et err, e')
        in
```

```
        let args' = List.map2 check_call fd.formals args
        in (fd.typ, SCall(fname, args'))
    | Index (name, ex) ->
        let (t, e) = expr ex in
        match t with
          Int -> (match (type_of_identifier name) with
            List(t') -> (t' , SIndex(name, (Int, e)))
            | _ -> raise ( Failure ("illegal type. expecting an array, found " ^
string_of_typ (type_of_identifier name))))
          | _ -> raise ( Failure ("illegal array index type. expecting Int, found "
^ string_of_typ t))
  in

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

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

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

```
        in SBlock(check_stmt_list sl)

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

Code Generation

codegen.ml

```
(* Code generation *)

module L = Llvm
module A = Ast
open Sast

(* translate : Sast.program -> Llvm.module *)
module StringMap = Map.Make(String)

let translate (globals, functions) =
 let context = L.global_context () in
 let llmem = L.MemoryBuffer.of_file "konig.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 "Konig" in

 (* Get types from the context *)
 let i32_t      = L.i32_type    context
 and i8_t       = L.i8_type     context
 and i1_t       = L.i1_type     context
 and float_t    = L.double_type context
 and void_t     = L.void_type   context
 and arr_t      = L.pointer_type (match L.type_by_name llm "struct.Array" with
     None -> raise (Failure "the array type is not defined.")
   | Some x -> x)
 and str_t      = L.pointer_type (L.i8_type context)
 and node_t     = L.pointer_type (match L.type_by_name llm "struct.Node" with
     None -> raise (Failure "the node type is not defined.")
   | Some x -> x)
 and edge_t     = L.pointer_type (match L.type_by_name llm "struct.Edge" with
   None -> raise (Failure "the node type is not defined.")
 | Some x -> x)
 and graph_t    = L.pointer_type (match L.type_by_name llm "struct.Graph" with
     None -> raise (Failure "the graph type is not defined.")
   | Some x -> x)
 and void_ptr_t = L.pointer_type (L.i8_type context)
```

```
in

(* Return the LLVM type for a MicroC type *)
let rec ltype_of_typ = function
    A.Int   -> i32_t
  | A.Bool  -> i1_t
  | A.Float -> float_t
  | A.Void  -> void_t
  | A.Edge  -> edge_t
  | A.Str -> str_t
  | A.List(typ)  -> arr_t
  | A.Node(typ)  -> node_t
  | A.Graph(typ) -> graph_t
in

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

(* 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
let print_node_t =
    L.function_type i32_t [| node_t |] in
let print_node_f =
    L.declare_function "print_node" print_node_t the_module in
let print_edge_t =
   L.function_type i32_t [| edge_t |] in
let print_edge_f =
   L.declare_function "print_edge" print_edge_t the_module in
let print_graph_t =
    L.function_type i32_t [| graph_t |] in
let print_graph_f =
    L.declare_function "print_graph" print_graph_t the_module in
```

```
(* list functions *)
let init_array_t =
    L.function_type arr_t [||] in
let init_array_f =
    L.declare_function "init_array" init_array_t the_module in
let append_array_t =
    L.function_type i32_t [| arr_t; void_ptr_t |] in
let append_array_f =
    L.declare_function "append_array" append_array_t the_module in
let get_array_t =
    L.function_type void_ptr_t [| arr_t; i32_t |] in
let get_array_f =
    L.declare_function "get_array" get_array_t the_module in
let pop_array_t =
    L.function_type i32_t [| arr_t; i32_t |] in
let pop_array_f =
    L.declare_function "pop_array" pop_array_t the_module in

(* node functions *)
let init_node_t =
    L.function_type node_t [| void_ptr_t |] in
let init_node_f =
    L.declare_function "init_node" init_node_t the_module in
let add_node_t =
    L.function_type graph_t [| node_t; graph_t |] in
let add_node_f =
    L.declare_function "add_node" add_node_t the_module in
let del_node_t =
    L.function_type graph_t [| node_t; graph_t |] in
let del_node_f =
    L.declare_function "del_node" del_node_t the_module in

(* edge functions *)
let set_edge_t =
    L.function_type edge_t [| graph_t; node_t; node_t; float_t |] in
let set_edge_f =
    L.declare_function "set_edge" set_edge_t the_module in
let set_dir_edge_t =
    L.function_type edge_t [| graph_t; node_t; node_t; float_t |] in
let set_dir_edge_f =
```

```
    L.declare_function "set_dir_edge" set_dir_edge_t the_module in
let get_edge_t =
    L.function_type edge_t [| graph_t; node_t; node_t |] in
let get_edge_f =
    L.declare_function "get_edge" get_edge_t the_module in
let del_edge_t =
    L.function_type edge_t [| graph_t; node_t; node_t |] in
let del_edge_f =
    L.declare_function "del_edge" del_edge_t the_module in
let update_edge_t =
    L.function_type edge_t [| graph_t; node_t; node_t; float_t |] in
let update_edge_f =
    L.declare_function "update_edge" update_edge_t the_module in

(* graph functions *)
let init_graph_t =
    L.function_type graph_t [||] in
let init_graph_f =
    L.declare_function "init_graph" init_graph_t the_module in
let neighbors_t =
    L.function_type arr_t [| graph_t; node_t |] in
let neighbors_f =
    L.declare_function "neighbors" neighbors_t the_module in
let visualize_graph_t =
    L.function_type i32_t [| graph_t; str_t |] in
let visualize_graph_f =
    L.declare_function "visualize_graph" visualize_graph_t the_module in

(* property getters *)
let get_node_val_t =
    L.function_type void_ptr_t [| node_t |] in
let get_node_val_f =
    L.declare_function "get_node_val" get_node_val_t the_module in
let get_node_id_t =
    L.function_type str_t [| node_t |] in
let get_node_id_f =
    L.declare_function "get_node_id" get_node_id_t the_module in
let get_edge_directed_t =
    L.function_type i1_t [| edge_t |] in
let get_edge_directed_f =
    L.declare_function "get_edge_directed" get_edge_directed_t the_module in
```

```
let get_edge_weight_t =
    L.function_type float_t [| edge_t |] in
let get_edge_weight_f =
    L.declare_function "get_edge_weight" get_edge_weight_t the_module in
let get_edge_id_t =
    L.function_type str_t [| edge_t |] in
let get_edge_id_f =
    L.declare_function "get_edge_id" get_edge_id_t the_module in
let get_array_length_t =
    L.function_type i32_t [| arr_t |] in
let get_array_length_f =
    L.declare_function "get_array_length" get_array_length_t the_module in
let get_graph_nodes_t =
    L.function_type arr_t [| graph_t |] in
let get_graph_nodes_f =
    L.declare_function "get_graph_nodes" get_graph_nodes_t the_module in
let get_graph_edges_t =
    L.function_type arr_t [| graph_t |] in
let get_graph_edges_f =
    L.declare_function "get_graph_edges" get_graph_edges_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 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 ((gtyp, e) : sexpr) = match e with
      SLiteral i  -> L.const_int i32_t i
    | SBoolLit b  -> L.const_int i1_t (if b then 1 else 0)
    | SFliteral l -> L.const_float_of_string float_t l
    | SStrLit s   -> L.build_global_stringptr s "str" builder
    | SNoexpr     -> L.const_int i32_t 0
    | SId s       -> L.build_load (lookup s) s builder
    | SListLit exps ->
      let arr = L.build_call init_array_f [||] "init_array" builder in
      let exps' = List.map (fun e -> expr builder e) exps in
      let typ = match exps with
        [] -> A.Void
      | _  -> (fst (List.hd exps))
      in
```

```
      let append e_val =
        let data = match typ with
          (* if we're storing an object, the result is already a pointer, so don't
need to malloc space for it *)
          A.Node(_) | A.Edge | A.Graph(_) | A.List(_) | A.Str -> e_val
        | _ -> let d = L.build_malloc (ltype_of_typ typ) "data" builder in
               ignore(L.build_store e_val d builder); d
        in
        let vdata = L.build_bitcast data void_ptr_t "vdata" builder in
        L.build_call append_array_f [| arr; vdata |] "append_array" builder
      in
      ignore(List.map (fun e -> append e) exps'); arr
    | SNodeLit exps ->
      let typ = (fst (List.hd exps)) in
      let data =
        let e_val = expr builder (List.hd exps) in
        let d = L.build_malloc (ltype_of_typ typ) "data" builder in
        ignore(L.build_store e_val d builder); d
      in
      let vdata = L.build_bitcast data void_ptr_t "vdata" builder in
      (L.build_call init_node_f [| vdata |] "init_node" builder)
    | SGraphLit exps -> L.build_call init_graph_f [||] "init_node" builder
    | SAssign (s, e) -> let e' = expr builder e in
                        ignore(L.build_store e' (lookup s) builder); e'
    | SIndex (s, e) ->
      let arr   = L.build_load (lookup s) s builder in
      let t     = ltype_of_typ gtyp in
      let idx   = expr builder e in
      let vdata = L.build_call get_array_f [| arr; idx |] "get_array" builder in (
      match gtyp with
        (* if we're accessing an object, the result is already a pointer, so don't
need to load it *)
        A.Node(_) | A.Edge | A.Graph(_) | A.List(_) | A.Str ->
          L.build_bitcast vdata t "data" builder
      | _ ->
          let data = L.build_bitcast vdata (L.pointer_type t) "data" builder in
          L.build_load data "data" builder)
    | SProp (e, prop) ->
      let cast_and_load ptr = (
        let typ = (L.pointer_type (ltype_of_typ gtyp)) in
        let data = L.build_bitcast ptr typ "data" builder in
```

```
        L.build_load data "data" builder)
      in
      let e' = (expr builder e) in (
      match (fst e, prop) with
        (A.Node(_), "val") ->
          let vdata = L.build_call get_node_val_f [| e' |] "get_node_val" builder
in
          cast_and_load vdata
      | (A.Node(_), "id") ->
          L.build_call get_node_id_f [| e' |] "get_node_id" builder
      | (A.Edge, "directed") ->
          L.build_call get_edge_directed_f [| e' |] "get_edge_directed" builder
      | (A.Edge, "weight") ->
          L.build_call get_edge_weight_f [| e' |] "get_edge_weight" builder
      | (A.Edge, "id") ->
        L.build_call get_edge_id_f [| e' |] "get_edge_id" builder
      | (A.List(_), "length") ->
        L.build_call get_array_length_f [| e' |] "get_array_length" builder
      | (A.Graph(_), "nodes") ->
        L.build_call get_graph_nodes_f [| e' |] "get_graph_nodes" builder
      | (A.Graph(_), "edges") ->
        L.build_call get_graph_edges_f [| e' |] "get_graph_edges" builder
      | _ -> raise (Failure ("ERROR: internal error, semant should have
rejected")))
    | SBinop ((A.Float,_ ) as e1, op, e2) ->
      let e1' = expr builder e1
      and e2' = expr builder e2 in
      (match op with
        A.Add     -> L.build_fadd
      | A.Sub     -> L.build_fsub
      | A.Mult    -> L.build_fmul
      | A.Div     -> L.build_fdiv
      | A.Equal   -> L.build_fcmp L.Fcmp.Oeq
      | A.Neq     -> L.build_fcmp L.Fcmp.One
      | A.Less    -> L.build_fcmp L.Fcmp.Olt
      | A.Leq     -> L.build_fcmp L.Fcmp.Ole
      | A.Greater -> L.build_fcmp L.Fcmp.Ogt
      | A.Geq     -> L.build_fcmp L.Fcmp.Oge
      | A.And | A.Or | A.Addnode | A.Delnode ->
          raise (Failure "internal error: semant should have rejected and/or on
float")
```

```
      ) e1' e2' "tmp" builder
    | SBinop (e1, A.Addnode, e2) ->
      let n = expr builder e1
      and g = expr builder e2 in
      L.build_call add_node_f [| n; g |] "add_node" builder
    | SBinop (e1, A.Delnode, e2) ->
      let n = expr builder e1
      and g = expr builder e2 in
      L.build_call del_node_f [| n; g |] "del_node" builder
    | SBinop (e1, op, e2) ->
      let e1' = expr builder e1
      and e2' = expr builder e2 in
      (match op with
        A.Add     -> L.build_add
      | A.Sub     -> L.build_sub
      | A.Mult    -> L.build_mul
      | A.Div     -> L.build_sdiv
      | A.And     -> L.build_and
      | A.Or      -> L.build_or
      | A.Equal   -> L.build_icmp L.Icmp.Eq
      | A.Neq     -> L.build_icmp L.Icmp.Ne
      | A.Less    -> L.build_icmp L.Icmp.Slt
      | A.Leq     -> L.build_icmp L.Icmp.Sle
      | A.Greater -> L.build_icmp L.Icmp.Sgt
      | A.Geq     -> L.build_icmp L.Icmp.Sge
      | _ -> raise (Failure "internal error: semant should have rejected")
      ) e1' e2' "tmp" builder
    | SUnop(op, ((t, _) as e)) ->
      let e' = expr builder e in
      (match op with
        A.Neg when t = A.Float -> L.build_fneg
      | A.Neg                  -> L.build_neg
      | A.Not                  -> L.build_not) e' "tmp" builder
    (* print functions *)
    | SCall ("print", [e]) | SCall ("printb", [e]) ->
      L.build_call printf_func [| int_format_str ; (expr builder e) |]
        "printf" builder
    | SCall ("printf", [e]) ->
      L.build_call printf_func [| float_format_str ; (expr builder e) |]
        "printf" builder
    | SCall ("printString", [e]) ->
```

```
      L.build_call printf_func [| string_format_str ; (expr builder e) |]
        "printf" builder
    | SCall ("printNode", [e]) ->
      L.build_call print_node_f [| (expr builder e) |] "print_node" builder
    | SCall ("printEdge", [e]) ->
      L.build_call print_edge_f [| (expr builder e) |] "print_edge" builder
    | SCall ("printGraph", [e]) ->
      L.build_call print_graph_f [| (expr builder e) |] "print_graph" builder
    (* edge functions *)
    | SCall ("setEdge", [g; n1; n2; w]) ->
      let g'  = (expr builder g)
      and n1' = (expr builder n1)
      and n2' = (expr builder n2)
      and w' = (expr builder w)
      in
      L.build_call set_edge_f [| g'; n1'; n2'; w' |] "set_edge" builder
    | SCall ("setDirEdge", [g; n1; n2; w]) ->
      let g'  = (expr builder g)
      and n1' = (expr builder n1)
      and n2' = (expr builder n2)
      and w' = (expr builder w)
      in
      L.build_call set_dir_edge_f [| g'; n1'; n2'; w' |] "set_dir_edge" builder
    | SCall ("getEdge", [g; n1; n2]) ->
      let g'  = (expr builder g)
      and n1' = (expr builder n1)
      and n2' = (expr builder n2)
      in
      L.build_call get_edge_f [| g'; n1'; n2' |] "get_edge" builder
    | SCall ("deleteEdge", [g; n1; n2]) ->
      let g'  = (expr builder g)
      and n1' = (expr builder n1)
      and n2' = (expr builder n2)
      in
      L.build_call del_edge_f [| g'; n1'; n2' |] "del_edge" builder
    | SCall ("updateEdge", [g; n1; n2; w]) ->
      let g'  = (expr builder g)
      and n1' = (expr builder n1)
      and n2' = (expr builder n2)
      and w' = (expr builder w)
      in
```

```
      L.build_call update_edge_f [| g'; n1'; n2'; w' |] "update_edge" builder
    | SCall ("append", [a; e]) ->
      let arr  = (expr builder a)
      and e' = (expr builder e)
      and typ = (fst e) in
      let data = (match typ with
        (* if we're storing an object, the result is already a pointer, so don't
need to malloc space for it *)
        A.Node(_) | A.Edge | A.Graph(_) | A.List(_) | A.Str -> e'
      | _ -> let d = L.build_malloc (ltype_of_typ typ) "data" builder in
             ignore(L.build_store e' d builder); d)
      in
      let vdata = L.build_bitcast data void_ptr_t "vdata" builder in
      L.build_call append_array_f [| arr; vdata |] "append_array" builder
    | SCall ("pop", [a; idx]) ->
      let arr = (expr builder a) in
      let i   = (expr builder idx) in
      L.build_call pop_array_f [| arr; i |] "pop_array" builder
    | SCall ("neighbors", [g; n]) ->
      let g'  = (expr builder g)
      and n' = (expr builder n)
      in
      L.build_call neighbors_f [| g'; n' |] "neighbors" builder
    | SCall ("viz", [g; p]) ->
      let g'  = (expr builder g)
      and p' = (expr builder p)
      in
      L.build_call visualize_graph_f [| g'; p' |] "visualize_graph" builder
    | SCall (f, args) ->
       let (fdef, fdecl) = StringMap.find f function_decls in
  let llargs = List.rev (List.map (expr builder) (List.rev args)) in
  let result = (match fdecl.styp with
                      A.Void -> ""
                    | _ -> f ^ "_result") in
       L.build_call fdef (Array.of_list llargs) result builder
   in

   (* LLVM insists each basic block end with exactly one "terminator"
      instruction that transfers control.  This function runs "instr builder"
      if the current block does not already have a terminator.  Used,
      e.g., to handle the "fall off the end of the function" case. *)
```

```
  let add_terminal builder instr =
    match L.block_terminator (L.insertion_block builder) with
Some _ -> ()
    | None -> ignore (instr builder) in
   (* Build the code for the given statement; return the builder for
     the statement's successor (i.e., the next instruction will be built
     after the one generated by this call) *)

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

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

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

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

    | SWhile (predicate, body) ->
  let pred_bb = L.append_block context "while" the_function in
  ignore(L.build_br pred_bb builder);

  let body_bb = L.append_block context "while_body" the_function in
  add_terminal (stmt (L.builder_at_end context body_bb) body)
    (L.build_br pred_bb);
```

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

  let merge_bb = L.append_block context "merge" the_function in
  ignore(L.build_cond_br bool_val body_bb merge_bb pred_builder);
  L.builder_at_end context merge_bb

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

  (* Build the code for each statement in the function *)
  let builder = stmt builder (SBlock fdecl.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.Float -> L.build_ret (L.const_float float_t 0.0)
    | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
in

List.iter build_function_body functions;
the_module
```

konig.ml

```
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: ./konig.native [-a|-s|-l|-c] [file.ko]" 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 (Prettyast.string_of_program ast)
| _ -> let sast = Semant.check ast in
  match !action with
    Ast     -> ()
  | Sast    -> print_string (Prettysast.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)
```

konig.c

```
#include "konig.h"

array* init_array() {
   array* arr = (array *) malloc(sizeof(array));
   arr->length = 0;
   arr->head = NULL;
   arr->tail = NULL;
   return arr;
}

int32_t append_array(array* a, void* data) {
   int32_t new_len = a->length + 1;

   elem* new_elem = (elem*) malloc(sizeof(elem));
   new_elem->data = data;
   new_elem->prev = a->tail;
   new_elem->next = NULL;

   if (a->length == 0) {
       a->head = new_elem;
   } else {
```

```
        a->tail->next = new_elem;
    }

    a->tail = new_elem;
    a->length = new_len;
    return 0;
}

int32_t delete_array(array* a, elem* e) {
    if (a->head == NULL || e == NULL )
        return 0;
    if (a->head == e)
        a->head = e->next;
    if (e->next != NULL)
        e->next->prev = e->prev;
    if (e->prev != NULL)
        e->prev->next = e->next;
    if (a->tail == e)
        a->tail = e->prev;
    a->length--;
    return 0;
}

void* get_array(array* a, int32_t index) {
    // printf("get_array\n");
    // printf("get_idx: %d %d\n", index, (a->length)-1);

    if (index > (a->length)-1) {
        fprintf(stderr, "ERROR: array index out of bounds: %d\n", index);
        exit(1);
    }

    elem* curr = a->head;
    while (curr && index) {
        index--;
        curr = curr->next;
    }

    return curr->data;
}
```

```c
int32_t pop_array(array *a, int32_t index) {
    if (index > (a->length)-1) {
        fprintf(stderr, "ERROR: array index out of bounds: %d\n", index);
        exit(1);
    }

    elem* curr = a->head;
    while (curr && index) {
        index--;
        curr = curr->next;
    }

    delete_array(a, curr);

    return 0;
}



node* init_node(void* data) {
    node* n =  (node *) malloc(sizeof(node));
    n->id = random_id(ID_LEN);
    n->data = data;
    // printf("node initialized successfully!\n");
    // printf("node id: %p\n", n);
    return n;
}

edge* init_edge(node* from, node* to, bool directed, double weight) {
    edge* e = (edge *) malloc(sizeof(edge));
    e->id = random_id(ID_LEN);
    e->from = from;
    e->to = to;
    e->directed = directed;
    e->weight = weight;
    return e;
}

graph* init_graph() {
    graph* g = (graph *) malloc(sizeof(graph));
    g->id = random_id(ID_LEN);
    g->nodes = init_array();
```

```
    g->edges = init_array();
    return g;
}

array* neighbors(graph* g, node* n) {
    char* ref = n->id;
    array* result = init_array();
    elem* curr = g->edges->head;
    while (curr) {
        edge* e = (edge*)curr->data;
        if (strcmp(ref, e->from->id) == 0 && find_elem_by_id(e->to->id, result) ==
NULL)
            append_array(result, e->to);
        if (strcmp(ref, e->to->id) == 0 && find_elem_by_id(e->from->id, result) ==
NULL)
            append_array(result, e->from);
        curr = curr->next;
    }
    return result;
}

elem* find_elem_by_id(char* id, array* nodes) {
    elem* curr = nodes->head;
    while (curr) {
        node* n = (node*)curr->data;
        if (strcmp(id, n->id) == 0)
            return curr;
        curr = curr->next;
    }
    return NULL;
}

int32_t find_index_by_id(char* id, array* a) {
    elem* curr = a->head;
    int32_t idx = 0;
    while (curr) {
        node* n = (node*)curr->data;
        if (strcmp(n->id, id) == 0)
            return idx;
        curr = curr->next;
        idx++;
```

```
    }
    return -1;
}

graph* add_node(node* n, graph* g) {
    if (find_elem_by_id(n->id, g->nodes) != NULL) {
        fprintf(stderr, "ERROR: adding duplicate node to graph");
        exit(1);
    }
    append_array(g->nodes, n);
    return g;
}

graph* del_node(node* n, graph* g) {
    elem* e = find_elem_by_id(n->id, g->nodes);
    delete_array(g->nodes, e);
    return g;
}

elem* find_elem_by_from_to(graph* g, node* from, node* to) {
    elem* curr = g->edges->head;
    while (curr) {
        edge* e = (edge*)curr->data;
        if (
            (strcmp(e->from->id, from->id) == 0 && strcmp(e->to->id,   to->id) == 0)
||
            (strcmp(e->to->id,   from->id) == 0 && strcmp(e->from->id, to->id) == 0)
        )
            return curr;
        curr = curr->next;
    }
    return NULL;
}

edge* set_edge_helper(graph* g, node* from, node* to, bool directed, double weight)
{
    elem* elem1 = find_elem_by_id(from->id, g->nodes);
    elem* elem2 = find_elem_by_id(to->id, g->nodes);

    if (elem1 == NULL || elem2 == NULL) {
        fprintf(stderr, "ERROR: attempting to create edge between nodes not in
```

```
graph\n");
        exit(1);
    }

    edge* e = init_edge(from, to, directed, weight);
    append_array(g->edges, e);
    return e;
}

edge* set_edge(graph* g, node* from, node* to, double weight) {
    return set_edge_helper(g, from, to, false, weight);
}

edge* set_dir_edge(graph* g, node* from, node* to, double weight) {
    return set_edge_helper(g, from, to, true, weight);
}

edge* get_edge(graph* g, node* from, node* to) {
    elem *el = find_elem_by_from_to(g, from, to);

    if (el)
        return (edge*)el->data;

    fprintf(stderr, "ERROR: attempting to access non-existing edge\n");
    exit(1);
}

edge* del_edge(graph* g, node* from, node* to) {
    elem *el = find_elem_by_from_to(g, from, to);

    if (el) {
        delete_array(g->edges, el);
        return (edge*)el->data;
    }

    fprintf(stderr, "ERROR: attempting to delete non-existing edge\n");
    exit(1);
}

edge* update_edge(graph* g, node* from, node* to, double weight) {
    elem *el = find_elem_by_from_to(g, from, to);
```

```
    if (el) {
        edge *e = el->data;
        e->weight = weight;
        return (edge*)el->data;
    }

    fprintf(stderr, "ERROR: attempting to update non-existing edge\n");
    exit(1);
}

int32_t get_array_length(array* a) {
    return a->length;
}


void* get_node_val(node* n) {
    return n->data;
}

char* get_node_id(node* n) {
    return n->id;
}

bool get_edge_directed(edge* e) {
    return e->directed;
}

double get_edge_weight(edge* e) {
    return e->weight;
}

char* get_edge_id(edge* e) {
    return e->id;
}

array* get_graph_nodes(graph* g) {
    return g->nodes;
}

array* get_graph_edges(graph* g) {
```

```
    return g->edges;
}

// Helper functions

char* random_id(int32_t length) {
    char* dest = (char*) malloc(sizeof(char) * (length + 1));
    char charset[] = "0123456789"
                     "abcdefghijklmnopqrstuvwxyz"
                     "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    int32_t i = 0;
    while (i < length) {
        int32_t index = (double) rand() / RAND_MAX * (sizeof charset - 1);
        dest[i] = charset[index];
        i++;
    }
    dest[length] = '\0';
    return dest;
}

int32_t print_node(node* n) {
    printf("node: {\n");
    printf("    id=\"%s\"\n", n->id);
    printf("}\n");
    return 0;
}

int32_t print_edge(edge* e) {
    printf("edge: {\n");
    printf("    id   =\"%s\",\n",e->id);
    printf("    from =\"%s\",\n",e->from->id);
    printf("    to   =\"%s\",\n",e->to->id);
    printf("    dir  = %s,\n",e->directed ? "true" : "false");
    printf("    w    = %f,\n",e->weight);
    printf("}\n");
    return 0;
}

int32_t print_graph(graph* g) {
    printf("graph: { id=\"%s\" }\n", g->id);
    printf("    %d nodes: {\n", g->nodes->length);
```

```
    int32_t i = 0;
    int32_t node_count = g->nodes->length;
    elem* curr = g->nodes->head;
    while (curr && i < node_count) {
        node* n = (node*)curr->data;
        print_node(n);
        curr = curr->next;
    }
    printf("    }\n");

    printf("    %d edges: {\n", g->edges->length);
    int32_t j = 0;
    int32_t edge_count = g->edges->length;
    curr = g->edges->head;
    while (curr && j < edge_count) {
        edge* e = (edge*)curr->data;
        print_edge(e);
        curr = curr->next;
    }
    printf("    }\n\n");
    return 0;
}

int32_t visualize_graph(graph* g, char* path) {
    if (IS_GRAPHVIZ_AVAILABLE) {
        return visualize_graph_helper(g, path);
    } else {
        fprintf(stderr, "WARNING: The Graphviz library was not compiled with Konig. "
\
                        "In order to use the viz() function, please install the " \
                        "Graphviz library and update the GRAPHVIZ_PATH variable " \
                        "in the \"./compile.sh\" script.\n");
        return -1;
    }
}
```

konig.h

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

#define ID_LEN 8

#ifndef IS_GRAPHVIZ_AVAILABLE
#define IS_GRAPHVIZ_AVAILABLE 0
#endif

// Type definitions

typedef struct Elem {
   void* data;
   struct Elem* next;
   struct Elem* prev;
} elem;

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

typedef struct Node {
   char* id;
   void* data;
} node;

typedef struct Edge {
   char* id;
   node* from;
   node* to;
   bool directed;
   double weight;
} edge;

typedef struct Graph {
```

```
   char* id;
   array* nodes;
   array* edges;
} graph;

array* init_array();
int32_t append_array(array* a, void* data);
int32_t delete_array(array* a, elem* e);
void* get_array(array* a, int32_t index);
int32_t pop_array(array *a, int32_t index);

node* init_node(void* data);
edge* init_edge(node* from, node* to, bool directed, double weight);
graph* init_graph();
array* neighbors(graph* g, node* n);

elem* find_elem_by_id(char* id, array* nodes);
int32_t find_index_by_id(char* id, array* a);
graph* add_node(node* n, graph* g);
graph* del_node(node* n, graph* g);

elem* find_elem_by_from_to(graph* g, node* from, node* to);
edge* set_edge_helper(graph* g, node* from, node* to, bool directed, double weight);
edge* set_edge(graph* g, node* from, node* to, double weight);
edge* set_dir_edge(graph* g, node* from, node* to, double weight);
edge* get_edge(graph* g, node* from, node* to);
edge* del_edge(graph* g, node* from, node* to);
edge* update_edge(graph* g, node* from, node* to, double weight);

int32_t get_array_length(array* a);
void* get_node_val(node* n);
char* get_node_id(node* n);
bool get_edge_directed(edge* e);
double get_edge_weight(edge* e);
char* get_edge_id(edge* e);
array* get_graph_nodes(graph* g);
array* get_graph_edges(graph* g);

char* random_id(int32_t length);
int32_t print_node(node* n);
int32_t print_edge(edge* e);
```

```
int32_t print_graph(graph* g);
int32_t visualize_graph(graph* g, char* path);
int32_t visualize_graph_helper(graph* g, char* path);
```

viz.c

```
#include "konig.h"
#include <graphviz/cgraph.h>
#include <graphviz/gvc.h>

int32_t visualize_graph_helper(graph* g, char* path) {

   Agraph_t* G;
   GVC_t* gvc;
   Agnode_t* agnodes[g->nodes->length];
   FILE *f;

   gvc = gvContext();
   G = agopen("graph", Agundirected, NULL);
   f = fopen(path, "wb");

   int32_t i = 0;
   elem* curr = g->nodes->head;
   while (curr) {
       node* n = (node*)curr->data;
       agnodes[i] = agnode(G, n->id, true);
       curr = curr->next;
       i++;
   }

   i = 0;
   curr = g->edges->head;
   agattr(G, AGEDGE, "label", "0");
   while (curr) {
       edge* e = (edge*)curr->data;
       char* weight = (char*) malloc(sizeof(char) * 10);
       snprintf(weight, 10, "%.3f", e->weight);
       int32_t from_idx = find_index_by_id(e->from->id, g->nodes);
       int32_t to_idx = find_index_by_id(e->to->id, g->nodes);
```

```
        Agedge_t *agE = agedge(G, agnodes[from_idx], agnodes[to_idx], "", true);
        agset(agE, "label", weight);
        curr = curr->next;
        i++;
        free(weight);
    }

    gvLayout (gvc, G, "dot");
    gvRender(gvc, G, "pdf", f);
    gvFreeLayout(gvc, G);
    agclose(G);
    fclose(f);
    return 0;
}
```

Other Files

demo.ko

```
ko int main() {

    node<string> n1;
    node<string> n2;
    node<string> n3;
    node<string> n4;
    graph<string> g;
    list<node<string>> nds;
    list<edge> ee;
    list<node<string>> nbrs;
    int i;

    // initialize graph
    g = new graph{};
    n1 = new node{"Matteo"};
    n2 = new node{"Delilah"};
    n3 = new node{"Lord"};
    n4 = new node{"Konig"};

    // add nodes to graph
```

```
    n1 @ g;
    n2 @ g;
    n3 @ g;
    n4 @ g;

    // add edges to graph
    setEdge(g, n1, n2, 1.1);
    setEdge(g, n2, n3, 1.2);
    setEdge(g, n3, n1, 1.3);
    setEdge(g, n4, n1, 1.4);
    setEdge(g, n4, n2, 1.5);
    setEdge(g, n4, n3, 1.6);

    nds = g.nodes;

    printString("These are the nodes in the graph:");
    for (i = 0; i < nds.length; i = i+1) {
        printString(nds[i].val);
        printNode(nds[i]);
    }

    ee = g.edges;

    printString("");
    printString("These are the edges in the graph:");
    for (i = 0; i < ee.length; i = i+1) {
        printEdge(ee[i]);
    }

    printString("");
    printString("These are the neighbors of Matteo:");

    nbrs = neighbors(g, n1);

    for (i = 0; i < nbrs.length; i = i+1) {
        printString(nbrs[i].val);
    }

    viz(g, "./demo.pdf");

}
```

Makefile

```
# Path to the LLVM compiler
LLC=/usr/local/opt/llvm/bin/llc
# Path to the C compiler
GCC=gcc
targets := $(wildcard **/*.mll) $(wildcard **/*.mly) $(wildcard **/*.ml)

.PHONY: all test ast_test sast_test clean
all: clean konig.native test

konig.native: $(targets)
   opam config exec -- \
   ocamlbuild -use-ocamlfind src/konig.native \
   -Is src/ast,src/sast -r \
   -package llvm -package llvm.analysis -package llvm.bitreader
   $(GCC) -c src/konig.c
   clang -emit-llvm -c src/konig.c -o konig.bc

test:
   python3 test.py

ast_test:
   ./konig.native -a ./src/ast/pretty_ast_test.ko

sast_test:
   ./konig.native -s ./src/sast/pretty_sast_test.ko

clean:
   ocamlbuild -clean
   rm -f *.o *.bc
```

compile.sh

```
#!/bin/bash

echo ""
echo "  ######################################"
echo "  #                                    #"
echo "  #   Welcome to the Konig compiler!   #"
echo "  #                                    #"
echo "  ######################################"
echo ""

# If for any reason you cannot install the graphviz library,
# change this USE_GRAPHVIZ to 0 and build Konig without it.
USE_GRAPHVIZ=1

GCC=gcc
LLC=/usr/local/opt/llvm/bin/llc
# If on Linux, change this to /usr/include/graphviz
GRAPHVIZ_PATH=/usr/local/opt/graphviz/lib
LIBS="-L$GRAPHVIZ_PATH -lgvc -lcgraph -lcdt"

if [ $(which llc) ]; then
   LLC=llc
fi

if [ "$#" -ne 1 ]; then
   echo "ERROR: incorrect number of parameters"
   echo "Usage:"
   echo "       ./compile.sh <input_file>"
   exit 1
fi

if [[ $USE_GRAPHVIZ -eq 1 ]] && [[ ! -d $GRAPHVIZ_PATH ]]; then
   echo "[!] WARNING: Konig cannot find the Graphviz library, so it will be built \
without the viz() function. If you'd like the viz() function to work, please \
install Graphviz, and update the GRAPHVIZ_PATH variable in the \"./compile.sh\" \
script."
   echo ""
fi
```

```
filename="$(basename -- $1)"
INPUT="${filename%.*}"

set -x

./konig.native -c $1 > "$INPUT.ll"
$LLC -relocation-model=pic $INPUT.ll > $INPUT.s

if [[ $USE_GRAPHVIZ -eq 1 ]] && [[ -d $GRAPHVIZ_PATH ]]; then
   $GCC -DIS_GRAPHVIZ_AVAILABLE=1 -c src/konig.c
   $GCC -DIS_GRAPHVIZ_AVAILABLE=1 -c src/viz.c
   $GCC -DIS_GRAPHVIZ_AVAILABLE=1 -o $INPUT.out $LIBS $INPUT.s konig.o viz.o
else
   $GCC -c src/konig.c
   $GCC -o $INPUT.out $INPUT.s konig.o
fi

rm $INPUT.s $INPUT.ll
```

Testing Files

test.py

```
import os
import subprocess

print("")
print("  ##########################################")
print("  #                                        #")
print("  #   Welcome to the Konig testing suite!  #")
print("  #                                        #")
print("  ##########################################")
print("")

TEST_DIR = "./test/"
tests = []

for f in os.listdir(TEST_DIR):
   if f.endswith(".ko"):
       tests.append(os.path.splitext(f)[0])

failed = 0

for test_file in tests:

   print("[+] Running test \"{}\"...".format(test_file))

   r1 = os.system("./compile.sh test/{}.ko > /dev/null".format(test_file))
   r2 = os.system("./{}.out &> test/{}.txt".format(test_file, test_file))
   res = subprocess.run(
       [
           "diff",
           "test/{}.txt".format(test_file),
           "test/{}.res".format(test_file)
       ],
       stdout=subprocess.PIPE,
       stderr=subprocess.PIPE)
   r3 = res.returncode

   if len(res.stdout) == 0 and r1 == 0 and r2 == 0 and r3 == 0:
```

```
        print("[+] test \"{}\" PASSED.".format(test_file))
    else:
        print("[!] test \"{}\" FAILED. (!!!)".format(test_file))
        failed += 1

    print("")

os.system("rm *.out test/*.txt")

print("")
if failed == 0:
    print("[+] {}/{} test cases PASSED".format(len(tests), len(tests)))
else:
    print("[!] {}/{} test cases FAILED (!!!)".format(failed, len(tests)))
```

test-add-edge.ko

```
ko int main() {
    node<int> n1;
    node<int> n2;
    graph<int> g;
    edge e;

    n1 = new node{100};
    n2 = new node{101};
    g = new graph{};
    n1 @ g;
    n2 @ g;
    e = setDirEdge(g, n1, n2, 123.45);
    printb(e.directed); // == true
    printf(e.weight); // == 123.45
}
```

test-add-edge.res

```
1
123.45
```

test-add-node.ko

```
ko int main() {
   node<int> n1;
   node<int> n2;
   node<int> n3;
   graph<int> g;

   n1 = new node{10};
   print(n1.val);
   n2 = new node{11};
   print(n2.val);
   n3 = new node{12};
   print(n3.val);
   g = new graph{};

   n1 @ g;
   n2 @ g;
   n3 @ g;
}
```

test-add-node.res

```
10
11
12
```

test-array-append.ko

```
ko int main() {
   list<node<int>> l;
   node<int> x;
   node<int> y;
   node<int> z;
   int i;

   l = [];
```

```
    x = new node{101};
    y = new node{202};
    z = new node{303};

    append(l, x);
    append(l, y);
    append(l, z);

    for (i=0; i < l.length; i = i+1) {
        print(l[i].val);
    }

}
```

test-array-append.res

```
101
202
303
```

test-array-literal.ko

```
ko int main() {
    list<int> x;
    x = [1, 2, 3];
    print(x[0]);
    print(x[1]);
    print(x[2]);
}
```

test-array-literal.res

```
1
2
3
```

test-array-pop.ko

```
ko int main() {
   list<node<int>> l;
   node<int> x;
   node<int> y;
   node<int> z;
   int i;

   l = [
       new node{101},
       new node{202},
       new node{303}
   ];

   for (i=0; i < l.length; i = i+1) {
       print(l[i].val);
   }

   pop(l, 1);

   for (i=0; i < l.length; i = i+1) {
       print(l[i].val);
   }

}
```

test-array-pop.res

```
101
202
303
101
303
```

test-del-edge.ko

```
ko int main() {
```

```
    node<int> x;
    node<int> y;
    edge e1;
    edge e2;
    graph<int> g;

    x = new node{100};
    y = new node{101};
    g = new graph{};

    x @ g;
    y @ g;

    e1 = setEdge(g, x, y, 0.99);
    e2 = deleteEdge(g, x, y);
    printb(e1.id == e2.id);
    printb(e1.directed == e2.directed);
    printb(e1.weight == e2.weight);
}
```

test-del-edge.res

```
1
1
1
```

test-del-node.ko

```
ko int main() {
    node<int> n1;
    node<int> n2;
    node<int> n3;
    graph<int> g;

    n1 = new node{10};
    n2 = new node{12};
    n3 = new node{13};
    g = new graph{};
```

```
   n1 @ g;
   n2 @ g;
   n1 ! g;
}
```

test-del-node.res (empty file)

test-get-edge.ko

```
ko int main() {
   node<int> x;
   node<int> y;
   edge e;
   graph<int> g;

   x = new node{100};
   y = new node{101};
   g = new graph{};

   x @ g;
   y @ g;

   setEdge(g, x, y, 0.99);
   e = getEdge(g, x, y);
   printb(e.directed);
   printf(e.weight);

}
```

test-get-edge.res

```
0
0.99
```

test-graph-edges.ko

```
ko int main() {
   graph<int> g;
   node<int> n1;
   node<int> n2;
   node<int> n3;
   list<edge> ee;
   int i;

   g = new graph{};
   n1 = new node{0};
   n2 = new node{1};
   n3 = new node{2};

   n1 @ g;
   n2 @ g;
   n3 @ g;

   setEdge(g, n1, n2, 1.0);
   setEdge(g, n1, n3, 1.0);
   setEdge(g, n2, n3, 1.0);

   ee = g.edges;

   printb(ee[0].id == getEdge(g, n1, n2).id);
   printb(ee[1].id == getEdge(g, n1, n3).id);
   printb(ee[2].id == getEdge(g, n2, n3).id);

}
```

test-graph-edges.res

```
1
1
1
```

test-graph-literal.ko

```
ko int main() {
```

```
   graph<int> g;
   g = new graph{};
}
```

test-graph-literal.res (empty file)

test-graph-nodes.ko

```
ko int main() {
   graph<int> g;
   node<int> n1;
   node<int> n2;
   node<int> n3;
   list<node<int>> nn;
   int i;

   g = new graph{};
   n1 = new node{0};
   n2 = new node{1};
   n3 = new node{2};

   n1 @ g;
   n2 @ g;
   n3 @ g;

   nn = g.nodes;

   printb(nn[0].id == n1.id);
   printb(nn[1].id == n2.id);
   printb(nn[2].id == n3.id);

}
```

test-graph-nodes.res

```
1
1
1
```

test-neighbors-func.ko

```
ko int main() {
   graph<string> g;
   node<string> n1;
   node<string> n2;
   node<string> n3;
   list<node<string>> nn;
   int i;

   g = new graph{};
   n1 = new node{"one"};
   n2 = new node{"two"};
   n3 = new node{"three"};

   n1 @ g;
   n2 @ g;
   n3 @ g;

   setEdge(g, n1, n2, 1.0);
   setEdge(g, n2, n1, 1.0);
   setEdge(g, n1, n3, 1.0);
   setEdge(g, n2, n3, 1.0);

   nn = neighbors(g, n1);

   print(nn.length);

   for (i = 0; i < nn.length; i = i+1) {
       printString(nn[i].val);
   }
}
```

test-neighbors-func.res

```
2
two
three
```

test-node-literal.ko

```
ko int main() {
   node<float> n;
   n = new node{12.345};
   printf(n.val);
}
```

test-node-literal.res

```
12.345
```

test-str-literal.ko

```
ko int main() {
   string x;
   node<string> y;
   list<string> z;



   x = "Hello world";
   y = new node{x};
   printString(x);
   printString(y.val);

   z = [
       "one",
       "two",
       "three"
   ];

   printString(z[0]);
   printString(z[1]);
   printString(z[2]);
}
```

test-str-literal.res

```
Hello world
Hello world
one
two
three
```

test-update-edge.ko

```
ko int main() {
   node<int> x;
   node<int> y;
   edge e1;
   edge e2;
   graph<int> g;

   x = new node{100};
   y = new node{101};
   g = new graph{};

   x @ g;
   y @ g;

   e1 = setEdge(g, x, y, 0.99);
   printf(e1.weight);
   updateEdge(g, x, y, 1.23);
   printf(e1.weight);
}
```

test-update-edge.res

```
0.99
1.23
```