Graphene

Graphing Algorithms Made Easy

Ashar Nadeem	an3056
Matthew Sanchez	mcs2307
Vasileios Kopanas	vk2398

Programming Language and Translators

Spring 2021

Table of Contents

1. White Paper

Graphene is an imperative programming language primarily used to implement graph algorithms in a quick and efficient manner. At its core, it is a subset of C11 with an expanded standard library featuring a multitude of very flexible data structures and methods.

Graphene abstracts from the user the need to define their own complex structs for every situation by providing its own handful of data structures that are very flexible and powerful in how they can be used. Behaving like classes in C++, they have many methods that can greatly reduce the amount of code needed to implement basic graph algorithms.

Graphene uses the C11 syntax,. and supports all of its basic arithmetic, logical, and relational operators as well as user-defined functions. In addition, the Graphene standard library features The language was inspired by aiming to mimic the pseudo code present in the CLRS Algorithms textbook, and implementing it as closely as possible but within the C11 syntax.

2. Tutorial

2.1 Compilation

To set up the compiler, simply run "**make**" from the root directory of the Graphene source code. This will generate the **graphene.native** file through which we can then compile and output the results of our Graphene code.

Graphene files use the **.gph** extension. Below we see two ways of compiling .gph files. The first will compile the program, spit out the binary, and then run the binary and display the output of the executable. The second, using the --**only-compile** flag, only compiles the binary and does **not** run the executable.

```
./graphene.sh program.gph
./graphene.sh program.gph --only-compile
```

2.2 Language Basics

Graphene aims to use a clean syntax similar to that of the C programming language, and inherits many of its features and design choices.

Below is an example of a very basic Graphene program. The program enters via the **main()** function and executes the printing of "Hello World!". As we can see, the syntax is nearly identical to C. Note that there is no return statement. Because LLVM insists on having terminators at the end of blocks, the compiler will automatically add said terminators (return statements), and thus in functions like main, returning values such as 0 (as is standard in C/C++) is redundant.

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

2.3 Primitive Data Types

Graphene supports 3 basic primitive data types. These are strings, integers and floating point numbers. These data types can be used across the language in other data structures or on their own, and support most arithmetic operations.

Below is a sample program that declares some strings, integers, and floating point numbers, assigns them values, and prints the results of mathematical operations conducted on them. In the case of strings, we simply print the value of the variable. Additionally, we see that these data types can be declared on the same line, like the floats are, as long as they are comma separated. They can also be assigned in-line with declaration, or separately.

```
int main(){

    int a = 10;
    int b = 5;
    float c, d;
    c = 0.5;
    d = 0.25;

    string e = "foo";
    print(e);

    print(a + b);
    print(c + d);

    print(a - b);
    print(c - d);

    print(a * b);
    print(c * d);

    print(a / b);
    print(c / d);
}
```

2.4 Conditional/Relational Operators

Conditionals and relational operators can be used in Graphene to support logic within the program. The two logical operators, **if** and **else**, can be used together to perform code based on the result of binary expressions. The relation operators are listed below:

Greater Than:	>
Greater Than or Equals:	>=
Less Than:	<
Less Than or Equals:	<=
Equivalent:	==
Not Equivalent:	!=

The below code showcases some very simple logic using conditional and logical operators. We declare some variables, and print different values based on the relational logic specified within the code.

```
int main(){

    int a = 10;
    int b = 20;

    if(a > 10){
        print(a);
    }

    else{
        print(b);
    }

    if(a != b){
        print(a + b);
    }
}
```

2.5 Loops

Graphene supports the usage of **for()** and **while()** loops in order to implement iterative logic, where a certain code block may need to be repeated an arbitrary amount of times. Both loops behave the same under the hood, with the declaration being the main difference. For loops allow the iterator to be incremented within the function call, whereas while loops need that logic to happen outside the function call.

The sample code below simply iterates until the value of variable **i** reaches a certain state, continuing to update the value of variable **a**. At the end, we print the value of variable **a**, and we can see that this has increased substantially due to the continuous addition within the loops.

```
int main(){

    int a = 5;

    int i = 0;
    while(i < 10){
        a = a + a;
        i = i + 2;
    }

    i = 0;
    for(i; i < 5; i = i + 2){
        a = a + a;
    }

    print(a);
}
```

2.6 Functions

Graphene allows functions to be declared and used anywhere within the file. Functions keep independent copies of their variables, and allow logic to be abstracted away from the main portions of the code. All functions are global and can be used at will anywhere within the program. While the **main()** function is required, others can be declared and used at will.

Let us explore the example below. We declare two functions, **foo()** and **bar()**, that take in some integers and do basic arithmetic. We then call them in our **main()** function, printing out the result.

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

int bar(int c){
    int d = c + 10;
    return d;
}

int main(){
    int sum = foo(2, 4);
    print(sum);
}
```

2.7 Data Structures

Where Graphene shines is its ability to make graph algorithms extremely simple with the usage of built in data structures. These include **lists**, **nodes**, **edges**, and **graphs**. Each of these data structures is rich with built in methods to manipulate data very easily. Each of the data structures supports all the primitive data types of Graphene, and thus are extremely flexible in how they can be used across a variation of algorithms and use cases.

2.7.1 Lists

Lists in Graphene are an abstract data type that contain a countable number of values. Lists can be declared as containing any data type, including other data structures. One of the neater parts of lists in Graphene is the ability for them to act as either; stacks, heaps, or queues; using the different types of built in methods. This greatly simplifies graph algorithms such as BFS/DFS, and allows greater flexibility when dealing with data. The built in methods and fields of lists are as follows:

- **[index]**
- **push_back()**
- **push_front()**
- **pop_back()**
- **pop_front()**
- **peep_back()**
- **peep_front()**
- **size**
- **add_all()**

In the program below, we see some of the powerful features of lists being displayed. Elements are easily pushed onto the end of the list through a loop, and we are able to access them through either indexing or "peeking" at the front or back. We can also check the size of the list, and pop elements from the front or back, giving us the functionality of stack or a queue.

```
int main(){

    list<int> l;

    int i = 0;
    while(i < 5){
        l.push_back(i);
        i = i + 1;
    }

    i = 0;
    while(i < l.size){
        print(l[i]);
        i = i + 1;
    }

    print(l.peek_front());

    while(l.size > 0){
        print(l.size);
        l.pop_front();
    }
}
```

2.7.2 Nodes

Nodes in Graphene are a built in data type that behave similar to structs in C. They are identical to nodes (also known as vertices) in graph theory. Nodes are differentiated by their unique integer ids (assigned by the user), and can contain data of any predefined primitive data type. The fields of nodes are listed below:

- **id**
- **val**
- **edges**

The following program shows some basic assignment of the id and value to be stored in our node. These can be overridden or assigned to other variables at any time, and allow our node data structure to hold data while being easily identifiable, when added to a graph data structure, to be explored later. Additionally, the **edges** field will be explored in the next section.

```
int main(){
    node<string> m;
    m.id = 1;
    m.val = "Terminal";

    print(m.id);
    print(m.val);
}
```

2.7.3 Edges

Graphene has the concept of edges, which, just like in graph theory, are data structures that connect together two different nodes, or the same node to itself. Edges between nodes can be directed or undirected, and can contain any value of a primitive data type, given that it matches that of the nodes. Edges have the following methods:

- **weight**
- **t**
- **dest**

As we can see below, we have declared two different nodes, and set some edges between them. The first edge, from **m** to **n**, is undirected. The second, from **n** to **m**, is directed, and has weight 5. We can index the edge of a node that return a list of said nodes. Additionally, we can access the weight of the edge, whether or not it is traversable, as well as getting the destination node.

```
int main(){

    node<int> m, n;

    m <-> n;
    n ->[5] m;

    list<edge<int>> l;
    l = n.edges;

    edge<int> e = l[1];
    print(e.weight);

    if(e.t == 1){
        print("Traversable");
    }
    node<int> p = e.dest;
}
```

2.7.4 Graphs

Graphs are data structures that specialize in holding nodes of any type, and make node operations on a list of nodes very easy. All nodes in a graph must be of the same type, as specified when declaring the graph itself. The methods of graphs are as follows:

- **contains()**
- **contains_id()**
- **add_node()**
- **add()**

We can declare a graph of a type as seen below, and add multiple nodes to it. Nodes that have been declared can be added, in addition to inline deceleration when adding them to the graph within the function call. We can find nodes in graphs by either their id, or by comparison.

```
int main(){

    graph<int> g;

    node<int> m, n, o;
    g.add_node(m);
    g.add_node(n);
    g.add_node(o);
    g.add(4, 5);

    if(g.contains(m)){
        if(g.contains_id(4)){
            print("Success");
        }
    }
}
```

2.8 Sample Use Case

As we learned in lecture, Shift Reduce parsing for a context free grammar can be done with just an automaton and some stacks. Graphene has both of these features, a graph with string edge types can be created to accurately reflect a LR(0) automaton, and lists can be used as the stack and input feed. The Graphene performs shift reduce parsing on the grammar below. The automaton graph is initialized in the init() function, and the rules of the grammar are represented in the reduce() function. All other functions are drivers for the parsing algorithm, are generalized to work for any grammar correctly represented with the graph and reduce().

0 $A' \to A$ with terminals x, y, and z and non-terminals A', A, and B.
1 $A \to B\,x\,A$
2 $A \to y$
3 $B \to$
4 $B \to z$

```
int main() {

    //g is the LR(0) automaton
    graph<string> g;

    //create the states
    g = init(g);

    list<string> input, stack;
    list<int> states;
    node<string> state;

    //input tokens
    input.add_all( "x", "z", "x", "y", "$");

    state = g[0];
    states.push_front(state.id);
    while(state.id != 6) {
        state = action(g, state, stack, states, input);
    }
    print("ACCEPT");
    print(state);

}

//performs the next step in SR parsing given a valid
automaton and grammar
node<string> action(graph<string> g, node<string> state,
list<string> stack,
            list<int> states, list<string> input) {

    print("top of stack:");
    if(stack.size) {
        print(stack.peek_front());
    }

    print("state: ");
    print(state.id);

    print("next input:");
    print(input.peek_front());

    string s = input.peek_front();
    string res;

    if(s == state.val) {

        stack.push_front(reduce(stack, states));
        state = g[states.peek_front()];
        state = traverse(state, stack.peek_front());
        states.push_front(state.id);
    }
    else {
        print("shift");
        state = traverse(state, input.peek_front());
        stack.push_front(input.pop_front());
        states.push_front(state.id);

    }
    print("");
    return state;
}

//generalized traversal
node<string> traverse(node<string> n, string val) {
    int i = 0;
    while(i < n.edges.size) {
        if(n.edges[i].t) {
            if(n.edges[i].weight == val) {
                return n.edges[i].dest;
            }
        }
        i = i + 1;
    }
    //segfault
    print("invalid transition, REJECT");
    return n.edges[i].dest;
}

//encoding of LR(0) automaton
graph<string> init(graph<string> g) {
    g.add(0, "x");
    g.add(1, "NONE");
    g.add(2, "$");
    g.add(3, "x");
    g.add(4, "x");
    g.add(5, "$");
    g.add(6, "$");

    //create transitions
    g[0] ->["A"] g[6];
    g[0] ->["B"] g[1] ->["x"] g[4];
    g[0] ->["y"] g[2];
    g[0] ->["z"] g[3];


    g[4] ->["B"] g[1];
    g[4] ->["y"] g[2];
    g[4] ->["z"] g[3];
    g[4] ->["A"] g[5];
    return g;
}

//encoding of grammar
string reduce(list<string> stack, list<int> states) {
    if(states.peek_front() == 6) {
        if(stack.peek_front() == "A") {
            stack.pop_front();
            states.pop_front();
            print("R0");
            return "A'";
        }
    }
    if(states.peek_front() == 5) {
        if(stack.peek_front() == "A") {
            stack.pop_front();
            states.pop_front();
        }
        else {
            print("reduce error, REJECT");
            return stack[stack.size];
        }
        if(stack.peek_front() == "x") {
            stack.pop_front();
            states.pop_front();
        }
        else {
            print("reduce error, REJECT");
            return stack[stack.size];
        }
        if(stack.peek_front() == "B" ) {
            stack.pop_front();
            states.pop_front();
            print("R1");
            return "A";
        }
    }
    if(states.peek_front() == 2) {
        if(stack.peek_front() == "y") {
            stack.pop_front();
            states.pop_front();
            print("R2");
            return "A";
        }
    }
    if(states.peek_front() == 0 || states.peek_front() ==
4) {
        print("R3");
        return "B";
    }
    if(states.peek_front() == 3) {
        if(stack.peek_front() == "z") {
            stack.pop_front();
            states.pop_front();
            print("R4");
            return "B";
        }
    }
    print("reduce error, REJECT");
    return stack[stack.size];
}
```

This code generates the following output:

```
output:
top of stack:
state:
0
next input:
x
R3

top of stack:
B
state:
1
next input:
x
shift

top of stack:
x
state:
4
next input:
z
shift

top of stack:
z
state:
3
next input:
x
R4

top of stack:
B
state:
1
next input:
x
shift
```

```
top of stack:
x
state:
4
next input:
y
shift

top of stack:
y
state:
2
next input:
$
R2

top of stack:
A
state:
5
next input:
$
R1

top of stack:
A
state:
5
next input:
$
R1

ACCEPT
id: 6, val: $, edges: 1
```

3. Language Manual

3.1 Data Types

3.1.1 Primitive Types

Integer

An integer is a sequence of decimal digits, always using decimal notation. We will be treating integers as booleans, similarly to how C already does, 0 denotes false, nonzero denotes true.
ex. 3

Float

A float consists of an integer part, a decimal point, followed by a fraction part. The integer and fraction parts both consist of a sequence of digits. None of these can be missing. Again, strictly decimal.
ex. 3.2

String

A string is a sequence of characters surrounded by double quotes. Strings are immutable. All strings are automatically appended with a null terminator
ex. "Hello World"

3.1.2 Built-in Types

The built-in types wrap other types and interact with each other to enable graph functionality. Wrapped types are denoted by <>, and when allowed, can be nested.

List

A list is a doubly-linked list that holds elements of a declared type. List data is dynamically allocated at the time of insertion, allowing for global scope. Lists can warp any type, including nodes, edges, and graphs. Lists are indexed using []

with an in argument. Lists have an accessible size field.

ex.	list<int> l ;

insert element:

```
        ex.     l.push_front(element) ;
                l.push_back(element) ;
```

remove and get element:

```
        ex.     l.pop_front() ;
                l.pop_back() ;
```

get element:

```
        ex.     l[0] ;
                l.peek_front() ;
                l.peek_back() ;
```

<u>Node</u>

A node contains an id of type int and a value of the declared type. Both the id and value can be accessed and altered. It also holds a list of edges, which can only be accessed. Nodes can only wrap int, float, and string types. The node data type stores a reference to a node, and this reference can be reassigned to other nodes freely.

ex.	node<int> n;

id access:

```
        ex.     n.id ;
```

val access:

```
        ex.     n.val ;
```

edges access:

```
        ex.     n.edges ;
```

<u>Edge</u>

An edge behaves as a struct and holds a weight of any primitive type, a destination node, whose type must match the edge's weight, and an integer indicating whether the edge is traversable. If traversable is 1, it means the edge is directed towards the node that it holds; else it is 0, indicating that the node whose edge list the edge is a member of has an incoming but untraversable edge from the "destination" node. Edges cannot be directly initialized by the user, edges are created using Graphene's edge operators and can only be used to store results of accessing a node's edgelist.

ex. edge<int> e = n.edges[0] ;

weight access:

ex. e.weight ;

traversability access:

ex. e.t ;

destination access:

ex. e.dest ;

Graph

A graph is a collection of nodes that hold values of a declared type. Nodes within the graph can be accessed using their ids. Nodes are accessed by their id using the [] operator.

ex. graph<int> g ;

index by node id:

ex. g[0] ;

node list (can be accessed and indexed as a regular list):

ex. g.nodes ;

3.2 Lexical Conventions

There are five kinds of tokens: keywords, identifiers, literals, expression operators, and control flow separators, similar to C. Spaces, tab characters, and newline characters are ignored aside from however they may separate tokens.

Comments

Multi-line comments: /* starts a comment, terminated by */

Single-line comments: // starts a comments, terminated by \n

Keywords

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

void	int	float	string
graph	node	edge	list
if	else	for	while
return			

Literals

These include the three primitive data types. Integers must consist entirely of decimal digits. Floats must consist of the decimal digits, a decimal, and the fractional value, with no leading zeros. Strings must be enclosed in double quotes.

Identifiers

Identifiers are strings that reference types, including function names and variables; variables can be declared or assigned freely, although types cannot be changed. Identifiers must start with an upper or lower case letter, and this can be followed by any amount of letters, digits, or underscores.

3.3 Expressions

Expressions are recursively built out of primary expressions and various operators, which are the following:

3.3.1 Primary Expressions

Identifier

An identifier is a primary expression, provided it has been suitably declared. Its type is permanently specified at its declaration, and it evaluates to its stored value. This stored value must be assigned before it is accessed.

Literal

An integer, float, or string literal.

(expression)

A parenthesized expression is a primary expression whose type and value are identical to those of the unadorned expression.

primary-expression (expression-list$_{optional}$)

A primary expression followed by parentheses containing a possibly empty, comma-separated list of expressions, is a function call. The primary expression must be of type *function*, and the result of the function call corresponds to the specified function type. Int and Float types are passed by value, String and all built-in types are passed by reference.

primary-expression . fieldname

An expression followed by a dot followed by the name of a member of a structure is a primary expression. '.' has precedence above all operators and is right associative.

primary-expression . method (expression-list$_{optional}$)

An id expression followed by a dot followed by the name of a function of its type followed by the arguments for the method. The id expression has to represent a special type since such functions are not supported in general.

3.3.2 Operators

Operators act on expressions, and require a sensical value of the expressions. All operators are listed in decreasing order of precedence.

Unary Operators

Expressions with unary operators group right-to-left

! *expression*

The result of the logical negation operator ! is 1 if the value of the expression is 0, 0 if the value of the expression is non-zero. The type of the result is int. This operator is applicable only to ints.

- expression

The result of the arithmetic negation operator the value of the int or float it is applied to, multiplied by -1.

Multiplicative Operators

The multiplicative operators *, /, and % group left-to-right, all have the same precedence.

*expression * expression*

The binary * operator indicates multiplication. Operands can be int or float, if at least one of them is float, the result will be float.

expression / expression

The binary / operator indicates division. Operands can be int or float, if at least one of them is float, the result will be float. Division by 0 will produce an error.

expression % expression

The binary % operator yields the remainder from the division of the first

expression by the second. Both operands must be int, and the result is

int. The remainder has the same sign as the dividend.

Additive

The additive operators + and – group left-to-right, all have the same precedence.

expression + expression

The result is the sum of the expressions. Operands can be int or float, if at least one of them is float, the result will be float.

expression – expression
The result is the difference of the operands. Operands can be int or float, but must be of the same type.

<u>Relational</u>
The relational operators group left-to-right, all have the same precedence.

expression < expression
Returns 0 if the expression is false and 1 if the expression is true.

expression > expression
Returns 0 if the expression is false and 1 if the expression is true.

expression <= expression
Returns 0 if the expression is false and 1 if the expression is true.

expression >= expression
Returns 0 if the expression is false and 1 if the expression is true.

expression == expression
The == (equal to) operator, can act on any two expressions of the same type. Compares by value for primitive types *including strings*, compares by reference for built-in types.

expression != expression
The != (not equal) operator, can act on any two expressions of the same type. Same comparison behavior as ==.

expression && expression
The && operator returns 1 if both its operands are non-zero, 0

otherwise. && guarantees left-to-right evaluation; moreover, the second operand is not evaluated if the first operand is 0. This operator is applicable only to ints.

expression || *expression*

The || operator returns 1 if either of its operands is non-zero, and 0 otherwise. || guarantees left-to-right evaluation; moreover, the second operand is not evaluated if the value of the first operand is non-zero. This operator is applicable only to ints.

id = *expression*

This assignment operator groups left-to-right. The value of the expression replaces that of the object referred to by the id. The operands need to have the same type. This applies to references of string and built-in types.

* Only == and != can be applied to strings and built-in types.

3.3.3 Built-in Functions & Special Operators

These are functions and operators built-in to the language, primarily to interact with built-in types.

list.push_front (*e*)

Adds the element to the beginning of the list. The element must be the same type declared to be contained by the list. Returns a reference to the list.

list.push_back (*e)*

Adds the element to the end of the list. The element must be the same type declared to be contained by the list. Returns a reference to the list.

list.pop_front (*e*)

Removes the element at the beginning of the list and returns it. Error if list is empty.

list.pop_back (*e*)

Removes the element at the end of the list and returns it. Error if list is empty.

list.peek_front (*e*)

Returns the element at the beginning of the list without removing it. Error if list is empty.

list.peek_back (*e*)

Returns the element at the end of the list and without removing it. Error if list is empty.

list.add_all(*e*, ...)

Adds one or more elements to the end of a list, in the order in which they are passed (left first). Returns a reference to the list.

nodeA ->[*weight*] *nodeB*

Adds a directed edge between nodeA and nodeB, with the defined weight. Weight can be of any primitive data type, but it must match the type wrapped by nodeA and nodeB.

nodeA -> *nodeB*

Adds a directed edge between nodeA and nodeB of default weight 0 for ints, 0.0 for floats, and “” for strings.

nodeA <->[*weight*] *nodeB*

Adds an undirected edge between nodeA and nodeB, with the defined weight. Weight type handled the same as for directed edges.

nodeA <-> *nodeB*

Adds an undirected edge between nodeA and nodeB of default weight, same default values as directed edges.

graph[*id*]

Returns the node with the specified id in the graph, if it exists.

graph.contains_id(*id*)

Returns 1 if this graph contains the node with the specified key. Returns 0 otherwise.

graph.contains(*node*)

Returns 1 if this graph contains this node reference, 0 otherwise.

graph.add_node(*node*)

Adds the specified node to this graph if it is not already present. The graph and node must contain the same type of value. Returns 1 if the new node is added, 0 otherwise.

graph.add(*key* , *value*)

Automatically constructs the node from key and value and adds the node to the graph. Type of value must match the graph type.

3.3.4 Programs

Programs consist of 0 or more of function declarations, and 0 or more variable declarations.

<u>Variable Declarations</u>

Form: *type id* ;

This is standard C, *type* denotes type and *id* will be the name of the variable.

<u>Function Declaration</u>

Form: *type id (formals_opt) { stmt_list }*
This is standard C, *type* denotes return type, *id* denotes the name of the function, *formals-opt* is an optional list of *formals,* of the form *type id*, and *stmt-list* denotes 0 or more statements (including variable declarations). There is a special void only to be used in function declaration where no value is returned. Each program requires a function named "main" to be declared.

<u>Types</u>
The types are
type:

int
float
string
node< *type** >
edge< *type** >
graph< *type** >
list< *type* >

*type** denotes only primitive types.

3.3.5 Statements

<u>Expression Statement</u>

Most statements are expression statements, which have the form

expression ;

Usually expression statements are assignments or function calls.

<u>Conditional Statement</u>

The two forms of the conditional statement are

if (*expression*) *statement ;*
if (*expression*) *statement* else *statement ;*

In both cases, *expression* is evaluated, and the following *statement* will be evaluated if it is true. If there is an else and *expression* evaluates to false, the *statement* following the else will be evaluated. The dangling else problem will be solved by connecting an else to the last elseless if.

While Statement

The while statement has the form

while (*expression*) *statement* ;

expression is evaluated prior to each iteration, the loop is terminated when it evaluates to false.*statement* is evaluated during each iteration.

For Statement

The for statement has the form:

for (*expression*$_{opt}$;*expression*$_{opt}$;*expression*$_{opt}$) *statement*

The first *expression* is evaluated upon entering the loop for the first time. The second *expression* is evaluated prior to each iteration, terminating the loop when evaluating to false. The third *expression* is evaluated after each iteration completes. *statement* is evaluated every iteration.

Return Statement

A function returns to its caller with a return statement, which has the form

return (*expression*) ;

The value of the *expression* is returned to the caller of the function, this must match the function return type.

Variable Declaration

form:

type id$_{list}$ *;*

type id = expression ;

Variable declarations are treated as statements. Multiple *id*s can be chained with commas to declare multiple variables of the same type at once. If one variable is declared it can also be assigned a value upon declaration

4. Project Plan

4.1 Development Process

4.1.1 Development

Tasks were often broken down and assigned to different members of the team and they were given ownership of that functionality. Tasks were aimed to be assigned so that a member could complete the end to end workflow without having any blockers or needing to wait for other team members. Additionally, lots of pair programming was used so that everyone on the team was on the same page and got insight into what others were working on and how they were approaching it. This also was great for debug sessions. Members were responsible for finishing all tasks assigned to them, and then testing it before pushing to Github, our version control system.

4.1.2 Testing

The team used test driven development to ensure a top quality finished result. Lots of integration tests were implemented for each and every feature of the language. Every time a team member worked on a feature, they were required to provide clear test cases, ideally both a success and failure scenario, that would cover the inner workings of the feature. Additionally, each push to Github required the team member to successfully run end to end integration tests, and only push up the code if all tests passed without errors. A similar procedure was followed when pulling code from the repo.

4.2 Programming Style Guide

The following coding standards were upheld during development:

- Lines of code should be no longer than 80 characters. Break lines when appropriate to achieve this standard

- Comment high level organizational features. This includes headers on large blocks of codes that might be working on one particular feature exclusively
- Provide variables consistent and self descriptive names. For example, “t” for types, “l” for lists, “e” for expressions
- Clearly indent pattern matching to increase the readability of functions
- Use 4 space indentation across all files
- Underscores are used to name variables rather than camelcase

4.3 Team Roles & Responsibilities

4.3.1 Ashar Nadeem: Manager & System Architect

Ashar served as the manager and system architect over the course of the project. As manager, Ashar was responsible for setting deadlines for deliverables and following through on the progress of team members. Ashar also took lead on the written portions of the project such as the proposal, language reference manual, and final report and glued together all the different components. Additionally, Ashar was also the system architect and helped keep all team members on the same page as to how the development environment was set up. This included making sure Docker was working on everyone's machines and the proper dependencies were included in the image.

4.3.2 Matthew Sanchez: Language Guru

Matthew served as the language guru and implemented most of the language features in OCaml. Matthew was responsible for managing all of the components of the compiler, making sure deadlines were met as far as deliverables, and making sure other members of the team were familiar with the code base. Matthew based all implementations off of the proposal and was able to successfully implement a vast majority of the initially specified features.

4.3.3 Vasilis Kopanas

Vasilis helped with finalizing the testing plan of the project, ensuring that the code worked as expected. The LRM was used as a reference when implementing sample blocks of code to test. Both success and failure test cases were created.

4.4 Software Development Environment

4.4.1 Communication

The primary form of communication for the team was through the **Discord** platform. Discord was chosen because of its simplicity, ease of use, and rich features. Teammates were able to utilize both text and voice chats, in addition to harnessing the applications screen sharing capabilities to pair-program when required.

4.4.2 Version Control

Github was used as a version control system to host the project code. Team members were easily able to push and pull code to and from the repository, and this proved beneficial when multiple people were working on different parts of the project. Team members were notified when code was changed so that everyone had the latest copy when making changes or updates to the codebase.

4.4.3 Text Editors

Each team member was free to use the text editor of their choice when making changes to the codebase. Some of the choices were **Visual Studio Code**, **Vim**, and **Nano**, as all three are readily available and easy to use on most operating systems. For project reports, language reference manuals, and proposals, **Google Docs** was used so that multiple team members could update and make changes together in real time.

4.4.4 Operating System

All code was compiled and tested on the **Ubuntu Focal** operating system. This dev environment was accessed using a **Docker** container that is based on the

MicroC compiler's Dockerfile, with additional dependency updates. The team decided to use Docker as it was very easy to give everyone the same exact coding environment, with the same dependencies, softwares, and tools already installed and working. This made development very easy and seamless, and the team did not run into any compatibility issues when developing.

4.4.5 Languages

The main source code for the Graphene programming language is in OCaml, in particular **OCaml 4.05.0**. Some of the graphing functionality and standard library features of Graphene are written in C, in particular **C11**. Other than the usage of these two for the source code, there are test suites and compilers written in bash scripts, and a Makefile for code compilation.

4.4.6 Compilers

Graphene needs two different compilers in order to be compiled. These are the **LLVM-10.0** compiler, as well as the **Clang-10** compiler for C. These dependencies are part of the development environment, and the Makefile will use these to generate the graphene.native and graphene.bc source files needed to compile .gph programs.

4.5 Project Timeline

```
Jan 20  Team formation

Jan 22  Finalize language idea and features

Feb 03  Submit project proposal

Feb 15  Start on scanner and parser

Feb 18  Meet with Arjun for proposal review

Feb 21  Start on language reference manual

Feb 23  Proofread language reference manual

Feb 24  Submit language reference manual and parser
```

```
Mar 09  Start on abstract syntax tree

Mar 11  Start on semantically checked abstract syntax tree

Mar 15  Start on semant and codegen

Mar 22  Compile “Hello World”

Mar 23  Create and start on test suite

Mar 24  Submit “Hello World”

Mar 29  Begin work on list functionality

Apr 04  Expand test suite

Apr 11  Expand other data structures

Apr 19  Start on final report

Apr 24  Finalize compiler code, run test suite

Apr 25  Proofread report, walkthrough presentation

Apr 26  Present to Edwards, submit to Courseworks
```

4.6 Git Log

```
commit 354746f22f261a5b17b660e58a980e9bacddd918
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Mon Apr 26 16:06:21 2021 -0400

    authorship

commit a8610de1d4c479404e2cbb22760ee61be4b1a4e0
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Mon Apr 26 14:11:44 2021 -0400

    naming convention
```

```
commit 276653d52bfced4aa66c467280a4758d47759133
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Mon Apr 26 14:10:56 2021 -0400

    fix merge conflict

commit c2e76a4802fc3c1270f3befbeeb5266d091a76c0
Merge: c6e7c04 648036f
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Mon Apr 26 12:06:21 2021 -0600

    Merge branch 'master' of
github.com:asharnadeem/graphene

commit c6e7c0491ed645584f1c907f98353ef0314789db
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Mon Apr 26 12:01:46 2021 -0600

    cleaned & finalized code, str-cmp, dup-decls, demo

commit 648036fc5630173fed1923a43cd873a6fc573761
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Mon Apr 26 13:53:51 2021 -0400

    formatting

commit dc1586332f59384aa53a8d238821b9adeebd5cce
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Mon Apr 26 13:05:06 2021 -0400

    more tests, gitignore

commit aa86371be6502e5613a0577b2f052ad1f633ca72
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Mon Apr 26 01:53:59 2021 -0400
```

```
    auto prettify

commit 12661995bcc6a3e9e2683399fcd80b7e1483bb14
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Mon Apr 26 01:52:30 2021 -0400

    code cleanup + prettify

commit 72e22efa7113e5d507178e8a6496b67d586b3dfa
Merge: 2176d1c daa578e
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Sun Apr 25 23:36:42 2021 -0600

    Merge branch 'master' of
github.com:asharnadeem/graphene

commit 2176d1c9ad057d945295590ede372f6428d6eb36
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Sun Apr 25 23:36:08 2021 -0600

    killed foreach

commit d1843decb5d7a2eb86215614f8e5cdb036114bf2
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Sun Apr 25 23:35:26 2021 -0600

    parser & scanner clean & finalized

commit daa578eab9d570292ebf31315ded62b02db65638
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Mon Apr 26 01:21:26 2021 -0400

    more tests, working bfs, makefile change, ignore .bc
file

commit 23f48c85e991ed654ed3240761ef28f40f0d72ea
```

```
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Sun Apr 25 21:20:12 2021 -0600

    edges can hold any type, everything works

commit 31cb12619385a13fe57cdfd9cc0de4e0e8eff255
Merge: cc296f9 542d5c3
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Sun Apr 25 18:24:27 2021 -0600

    Merge branch 'master' of
github.com:asharnadeem/graphene

commit cc296f98ef407aea98108b9b9f1d274715785352
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Sun Apr 25 18:21:51 2021 -0600

    fixed issue with pointers as lists elements!

commit 542d5c366b61b5860e3c0e0f30ae3566a9cdac98
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Sun Apr 25 19:01:50 2021 -0400

    add sast back and remove sasttest

commit 1ba809bd5d8aeaa3dab84d3c69e8631ba73d80dc
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Sun Apr 25 18:43:01 2021 -0400

    removed uneeded files and rename some stuff

commit e355a663da55298e846e365ced1be4bd17ad951c
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Sun Apr 25 18:20:49 2021 -0400

    more test files for report
```

```
commit 8b3edf8b8a9cfdc9ea7119f4505916bb062b3c74
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Sun Apr 25 11:00:54 2021 -0600

    rm graph.add_all (redundant)

commit fab021886fdbcb1a847888c439171cbafb007d33
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Sun Apr 25 10:47:22 2021 -0600

    list.add_all & fixed list indexing on pointer type

commit 40d2ef58fba6a5ab84fff4e6f3fe6798fee1971a
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Sun Apr 25 09:37:25 2021 -0600

    graph contains() and containsid() functions

commit 0e5f3786c849b0a1d2ca23adf6ccc57647e7377c
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Sun Apr 25 08:55:42 2021 -0600

    added list peek functions

commit 6c4f511c4ad7a12d2e08b69d6f00ce418f4ea64c
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Sun Apr 25 00:40:43 2021 -0600

    Neq for pointer types

commit 5f9417829f68c3e77f9635ef70d5d4545231c93a
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Sun Apr 25 00:26:52 2021 -0600

    int & float values work as boolean preds
```

```
commit 87b97c704135d0bf45df9ef861ba27b4236a96e2
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Sun Apr 25 00:00:09 2021 -0600

    graph.add function and pointer comparison

commit a23aa60d4bd2e05bddddfe291125a477023cb4db
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Sat Apr 24 20:50:43 2021 -0600

    moved all built-in funcs to generalized ast types

commit c178023fcead381cec00ed0ec4a92a5a3baa7db1
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Sat Apr 24 20:01:42 2021 -0600

    added universal print for primitivess & nodes

commit 0721f73c9bcd70c6e79e83fb2a0f41353492f5ca
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Sat Apr 24 18:02:41 2021 -0600

    fixed error in node.val storage

commit 17124e168690731b48e2278420259708d667b2b1
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Sat Apr 24 17:34:52 2021 -0400

    compiler for program

commit f897af8db4036bb7c21cbf3a8f8ad33bdd277591
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Sat Apr 24 17:34:34 2021 -0400

    added sample programs for final report
```

```
commit ba9ae9ea8866a9fa28dc533c0bf797a5f1a72a8c
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Sat Apr 24 13:51:18 2021 -0600

    todo update

commit 2ba3fe0d27686bb42c73e3c2a92ec12eb34777f7
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Sat Apr 24 13:50:50 2021 -0600

    todo update

commit fbef28db4681b5983b64883e22f236913aaf1271
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Sat Apr 24 13:49:04 2021 -0600

    added strings, fixed parsing DOT, cleaned indexing

commit 86e48bfaa40900659b2c72d4a169c2a7317e2500
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Sat Apr 24 10:46:24 2021 -0600

    todo

commit 90f15e2e754ce48286f1bfad8d6647a5e3253b82
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Sat Apr 24 10:44:03 2021 -0600

    completed struct field access for all structs

commit 87cecc10851b6d010a6219ce67b688f46b4ef5d8
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Fri Apr 23 15:10:20 2021 -0600

    addes list.size, fixed count err in push_back
```

```
commit 869c50519aa651d08bec6d6add2037ec48d3ebe8
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Fri Apr 23 14:47:07 2021 -0600

    checking for valid wrapped types

commit 83352b100b3729d30f918272e9b1a520c75227d3
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Fri Apr 23 14:02:49 2021 -0600

    push_back generalized beyond int

commit 573adb7c130907299eb6d72f9872b6e4760f8d7a
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Fri Apr 23 13:23:21 2021 -0600

    can now chain edge operators

commit 4c4af1df8f53320cc3bda73f5651754fb5985f4c
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Fri Apr 23 13:09:32 2021 -0600

    many vars can be declared (NOT INIT) at once

commit 825f7e1462debdeedaff8d2d93fe03099bfde2c6
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Thu Apr 22 23:17:30 2021 -0600

    list indexing now ast type, turned off ocaml warns

commit ad7bdc536892e03f3b0f977d54dec21bd5487ea4
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Thu Apr 22 19:52:34 2021 -0600

    fixed parser, objs generalized to expr not just ID
```

```
commit 1ba6549c060a0c151617145f3dc5273cc3f2ae77
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Tue Apr 20 19:13:39 2021 -0400

    added pops, fixed seg faults

commit 3881d12daaaa4072a8f291c1d6d6734f52727fee
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Sun Apr 18 17:40:24 2021 -0400

    list empty, get node, fixed add node

commit cac28c118661dd462e17d5fbb13239b7847368c6
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Sat Apr 17 18:34:43 2021 -0400

    graph working with add_node

commit 7f718e1358160e4c8ea87e5c689426f00d912ee1
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Fri Apr 16 10:07:15 2021 -0600

    killed printbig :(

commit 508da93e72e31ca61d843d651055d070f72bbd26
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Fri Apr 16 09:54:42 2021 -0600

    Every file in test suite compiles to executable

commit 45abac1568920800ac49184e98cd6e65ad2b61c9
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Fri Apr 16 01:36:44 2021 -0600

    major progress on lists, nodes, edges (void *)
```

```
commit 862c1c490a0d6107c3363c58834ed0677dcc0db0
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Thu Apr 15 18:58:24 2021 -0600

    node partially moved to c, everything still works

commit 81621c2b1c73fab95868506a7ae4cfc724c1e21a
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Tue Apr 13 13:06:49 2021 -0600

    list push_back and [] working!

commit 7572c3dd8df91ec417d73b1d21af17a344c45fd6
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Tue Apr 13 00:39:56 2021 -0400

    dockerfile with clang

commit 3d6508e0b2037c1b563229ad5a35765127b08c8d
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Tue Apr 13 00:30:08 2021 -0400

    comment out indexing

commit 0f522b7a79016cc5782541f07e07e4fdb1d5ad3a
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Mon Apr 12 11:57:10 2021 -0400

    list basic impl (index does NOT work)

commit 4e2ddf7b126801fffe7e8bfe4d7b739b7c14835b
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Thu Apr 8 19:57:10 2021 -0600

    adding sasttest.ml
```

```
commit 2aed8d666f958f4a3c199d96957e4aa74706e3a8
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Wed Apr 7 14:46:51 2021 -0600

    node<float> works (fixed floats in parser too)

commit 2871a5959cb184ae7d6e4aa32d77fb775dd0926e
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Tue Apr 6 23:14:50 2021 -0600

    node<int> working, using basic struct for now

commit 69188eac7b202299087352bd359d182da823ec67
Merge: 34b3cfa be52002
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Fri Mar 26 00:18:56 2021 -0600

    Merge branch 'master' of
github.com:asharnadeem/graphene

commit 34b3cfa7aee7aa8b35364761b9dffb337c86460a
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Fri Mar 26 00:17:46 2021 -0600

    added graph operators - works through semant

commit be520022b076a5cef9e59cd93ca2db5ca8c09bb9
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Wed Mar 24 16:55:49 2021 -0400

    more tests

commit 42dcc6efd920d782b3fa284b117b5821b869e55b
Merge: c38cc29 d8453c9
Author: matthew-sanchez <mcs2307@columbia.edu>
```

```
Date:   Wed Mar 24 14:28:44 2021 -0600

    Merge branch 'master' of
github.com:asharnadeem/graphene

commit c38cc29b9d1c7d11fd228f6bdc4e75c10f8a35f0
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Wed Mar 24 14:27:58 2021 -0600

    fixed vdecls - issue in parser

commit d8453c9e716b8797fc62d55c41d891ef1c16a4c5
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Wed Mar 24 16:09:59 2021 -0400

    more tests and makefile fix

commit 29f8ad12184cf1ab77ad7cd282cd5ea2faee4e4d
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Wed Mar 24 00:50:01 2021 -0400

    deleted error file

commit bfe4ac9f6126ef9148ab71692132878b072e4bb7
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Wed Mar 24 00:48:44 2021 -0400

    test cases, semant fixes, and makefile

commit db67a4c1379e1a7934ccdfe77f02c3d4da55b6d4
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Sun Mar 21 23:31:33 2021 -0600

    re-added decls and assign without locals

commit c20a12f55d5e6a1c5b38317303e7aa8064f02f4f
```

```
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Sun Mar 21 21:48:31 2021 -0400

    readded missing parser

commit 8b48645a64ffa5cf1ffaab0aee6b3ef1e27ec194
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Thu Mar 18 23:18:05 2021 -0400

    codegen + graphene.ml skeletons

commit 1ba150b5b7ea7fcb751e836c2a618b1f5d55e1bb
Author: matthew-sanchez
<79229258+matthew-sanchez@users.noreply.github.com>
Date:   Wed Mar 17 22:38:16 2021 -0600

    Delete ast.mli

    .mli is incorrect

commit b0c42b69f5f701d7e6f53360177dcd146cebae3d
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Wed Mar 17 22:36:05 2021 -0600

    semant.ml checks sast, works with updated tests

commit 451e387f187e6c6cf591884f49c610a6f0ae5a0f
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Wed Mar 17 15:20:30 2021 -0600

    sast.ml created, includes pretty print, not tested

commit a7f6c72c5b5661ab7a870863dd727a81b1aeb364
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Wed Mar 17 10:07:55 2021 -0600
```

```
    declaration + assignment now works

commit 2291a4c41dcb057b2040875001dff7461044a92a
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Wed Mar 17 09:49:53 2021 -0600

    else now works

commit 80155a1a13b04c6aa32461aed58a06a2d74ff4f4
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Tue Mar 16 01:35:11 2021 -0600

    updated test for all vars/types

commit 247b561d98823004c482b5aecae0120704e2d16b
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Tue Mar 16 00:02:47 2021 -0600

    fixed typo

commit 8508fbb174a4baa90781379b1aa76f2f3fbfe2be
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Tue Mar 16 00:00:40 2021 -0600

    removed accidental .swp

commit a4bced3e978e47e676cbf209f445c43a10193d85
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Mon Mar 15 23:59:52 2021 -0600

    start of tests folder

commit 670e22f0110a0307e6e32f5aa0c3ee66866690c2
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Mon Mar 15 22:45:24 2021 -0600
```

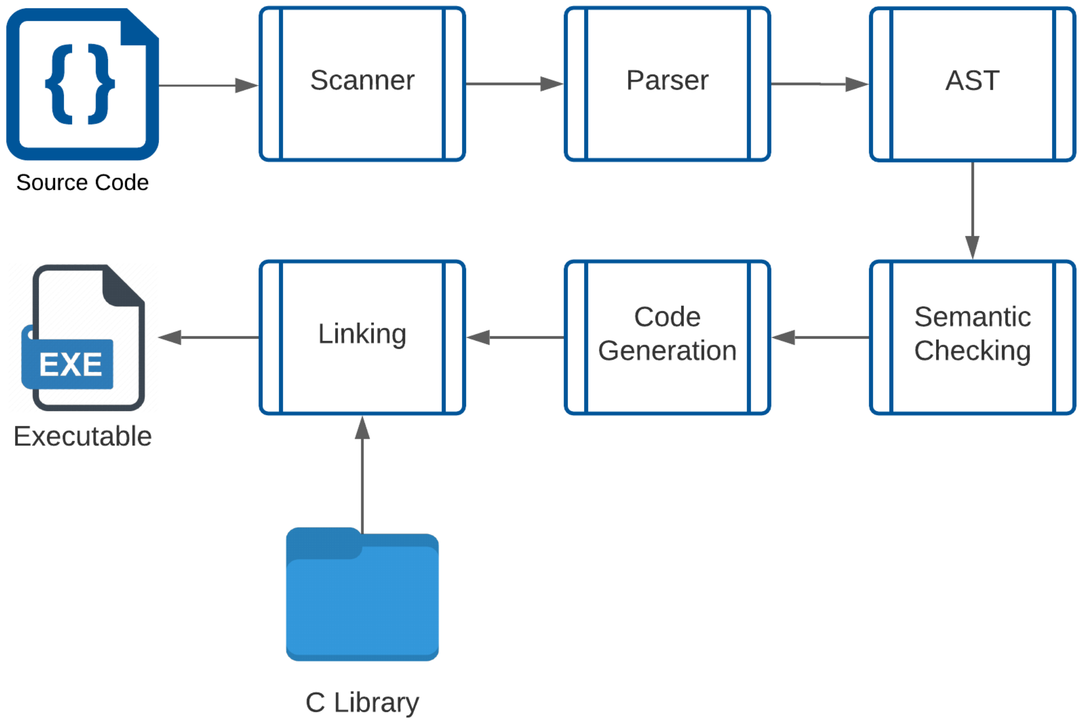
```
    confirmed working up through ast

    asttest pretty-prints input file

commit fd8fcc9184388ee7302ba10ce2dfda11f1813950
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Mon Mar 15 20:25:36 2021 -0600

    cleaned up ast, pretty-print, reduced parser

commit 8743eea6c9c81b993234ec6d344ef4e3a1529d57
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Mon Mar 15 13:54:59 2021 -0600

    added "int x = 1" type statements

commit 1d41ecbb66ceb4cc320a9325779b7a12a7f9b1e7
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Wed Feb 24 20:20:25 2021 -0700

    fixed vdecl

commit 88375e072937ec8a25108c1025bde9a710beaabd
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Wed Feb 24 21:41:47 2021 -0500

    accessing

commit 69fbf425e0a591b1d54e8e1b353a04fd61becc5b
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Wed Feb 24 19:34:43 2021 -0500

    edges

commit ceb670e7967f94333550a07dcfb7d72f7ba255b9
Author: Ashar Nadeem <an3056@columbia.edu>
```

```
Date:   Wed Feb 24 19:28:59 2021 -0500

    parser cleanup

commit be835050d2fe9182b5fce4bb456f8897a303ed0f
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Wed Feb 24 17:01:48 2021 -0700

    added mod

commit 4553f4203bfd2bfb909cb3e66e9a6ee1bfa48b02
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Wed Feb 24 17:40:05 2021 -0500

    fixed method call

commit c4acc0ce6f7feb8a57807b5487b5b1b4ad2fd779
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Wed Feb 24 17:24:33 2021 -0500

    foreach inclusion

commit a1f482ac64d0ea43bae1ada091d3abdca3d8b34a
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Wed Feb 24 17:16:11 2021 -0500

    node decleration

commit 796807b09c607b67acfbcf5b9dce2ea4c3259f54
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Wed Feb 24 01:12:11 2021 -0500

    break/continue, graph functions, index and arraylit

commit b2ff8e83954ce8e7314915fc5ab24ba47df071ef
Author: Ashar Nadeem <an3056@columbia.edu>
```

```
Date:   Tue Feb 23 14:57:15 2021 -0500

    adding edges parsing

commit 799ebf7d463bf5567f7f705e7a2e13e08b6610ee
Author: matthew-sanchez <mcs2307@columbia.edu>
Date:   Tue Feb 23 00:24:53 2021 -0700

    MCS: parser 99% done, complies, scanner updated.

commit 315f04dee05b84dbd70fb75f49a3c0f02180538b
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Sun Feb 21 19:25:13 2021 +0000

    scanner token changes, test parser, makefile *

commit e088107a4033e53ebe3f8a5eb7d3b9386282f0cd
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Sun Feb 21 18:31:03 2021 +0000

    makefile for scanner

commit 8428ec1ab43af7776cde64cd279260191fc9e1c4
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Sun Feb 21 13:23:02 2021 -0500

    init commit from matt

commit f9dd5b3d40de21d31135feb5e9360a38e18cc601
Author: Ashar Nadeem <an3056@columbia.edu>
Date:   Wed Feb 17 13:23:59 2021 -0500

    Initial commit
```

5. Architectural Design

5.1 Compiler Diagram



5.2 Source Code

The source code is the raw Graphene file (**.gph** extension) that the user writes and would like to compile. This source code is then read by the Graphene compiler and packaged into a binary executable that the user can then run at will.

5.3 Scanner

The scanner, written in **OCamlLex**, is used to generate tokens as specified in the lexical conventions. All whitespace and comment blocks are thrown away, and the input from the source code is tokenized and spat out to the parser. If the code contains any illegal sequence of characters that cannot be tokenized, the scanner will raise an error and fail the compilation of the program.

5.4 Parser

The parser, written in **OCamlYacc**, is used to generate an abstract syntax tree of the code. It works together with the ast.ml file to define the grammar rules, and serves to ensure that the sequence of tokens generated by the scanner is correct according to the grammar rules. In case of any errors, the parser will raise a parse error and fail compilation.

5.5 Semantic Checking

The semant, written in **OCaml**, goes through the abstract syntax tree and produces a semantically checked abstract syntax tree. The validity of our abstract syntax tree is checked, and syntax errors are raised with detailed messages to the user describing what went wrong should the semant fail the source code. Here, things such as variable declaration, type matching, and missing statements are validated.

Built in functions and field accessing are enabled through specific types in the ast, allowing for non-specific arguments/return types and providing the illusion of these object oriented features.

5.6 Code Generation

The codegen, written in **OCaml**, uses post order traversal to go through the semantically checked abstract syntax tree and generate the corresponding LLVM code. Most functionality is built into the codegen, and some is provided through linked C libraries to be explained further in the next section. Built-in LLVM functions were used in the codegen to pass and cast pointers between OCaml and C. Some methods had OCaml using malloc itself to allocate space and store values into memory.

5.7 C Library

The C library, written in **C11**, was where all of the data structures were implemented for graph algorithms. Most of them used linked list implementations for traversals, access, or indexing. Additionally, some data structures like lists did use the C library to implement methods such as push() or pop(). This C library linked to the codegen to provide support to many of the core features of the Graphene language.

6. Test Plan

6.1 Graphene to LLVM

Below is a sample program written in Graphene. The program declares a handful of nodes, assigns them id's, connects them to each other via edges, and then runs breadth-first search on the tree and finds a target node, printing out its metadata.

This sample program was chosen as it shows off most of the functionality of the Graphene programming language. All the data structures are used, with many of the methods of each being called at some point in the program.

```
node<int> bfs(node<int> root, node<int> target){

    graph<int> visited;
    node<int> origin, destination;

    list<node<int>> queue;
    queue.push_back(root);

    int i = 0;
    while(queue.size > 0){

        origin = queue.pop_front();

        if(origin == target){
            print(origin);
            return origin;
        }

        for(i = 0; i < origin.edges.size; i = i + 1){
            if(origin.edges[i].t){

                destination = origin.edges[i].dest;
                if(visited.contains(destination) == 0){
                    visited.add_node(destination);
                    queue.push_back(destination);
                }
            }
        }
    }
    return root;
}

int main(){

    node<int> m, n, o, p, q, r;
    m.id = 1;
    n.id = 2;
    o.id = 3;
    p.id = 4;
    q.id = 5;
    r.id = 6;

    m <-> n;
    m <-> p;
    n <-> o;
    n <-> p;
    p <-> q;
    q <-> r;

    bfs(m, r);
}
```

Below is the LLVM generated code for our **bfs.gph** file.

```
; ModuleID = 'Graphene'
source_filename = "Graphene"

%struct.list = type { i32, %struct.list_element* }
%struct.list_element = type { i8*, %struct.list_element* }
```

```
%struct.node = type { i32, i8*, %struct.list* }
%struct.edge = type { i8*, %struct.node*, i32 }
%struct.graph = type { %struct.list*, %struct.node* }

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%g\0A\00"
@fmt.2 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.3 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.4 = private unnamed_addr constant [4 x i8] c"%g\0A\00"
@fmt.5 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.6 = private unnamed_addr constant [28 x i8] c"id: %d, val:
%d, edges: %d\0A\00"

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

declare %struct.list* @list_init()

declare %struct.node* @node_init()

declare %struct.edge* @edge_init(i8*, %struct.node*, i32)

declare %struct.graph* @graph_init()

declare i8* @list_index(%struct.list*, i32)

declare i32 @list_push_back(%struct.list*, i8*)

declare i32 @list_push_front(%struct.list*, i8*)

declare i8* @list_pop_back(%struct.list*)

declare i8* @list_pop_front(%struct.list*)

declare i8* @list_peek_back(%struct.list*)

declare i8* @list_peek_front(%struct.list*)

declare i32 @graph_add_node(%struct.graph*, %struct.node*)
declare %struct.node* @graph_add(%struct.graph*, i32, i8*)

declare %struct.node* @graph_get_node(%struct.graph*, i32)

declare i32 @graph_contains_node(%struct.graph*, %struct.node*)

declare i32 @graph_contains_id(%struct.graph*, i32)

define i32 @main() {
entry:
  %m = alloca %struct.node*
  %n = alloca %struct.node*
  %o = alloca %struct.node*
  %p = alloca %struct.node*
  %q = alloca %struct.node*
  %r = alloca %struct.node*
  %init_node = call %struct.node* @node_init()
  store %struct.node* %init_node, %struct.node** %r
  %struct.ptr = getelementptr inbounds %struct.node,
%struct.node* %init_node, i32 0, i32 1
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i32*
getelementptr (i32, i32* null, i32 1) to i32))
  %nodeval = bitcast i8* %malloccall to i32*
  %cast = bitcast i32* %nodeval to i8*
  store i8* %cast, i8** %struct.ptr
  %init_node1 = call %struct.node* @node_init()
  store %struct.node* %init_node1, %struct.node** %q
  %struct.ptr2 = getelementptr inbounds %struct.node,
%struct.node* %init_node1, i32 0, i32 1
  %malloccall3 = tail call i8* @malloc(i32 ptrtoint (i32*
getelementptr (i32, i32* null, i32 1) to i32))
  %nodeval4 = bitcast i8* %malloccall3 to i32*
  %cast5 = bitcast i32* %nodeval4 to i8*
  store i8* %cast5, i8** %struct.ptr2
  %init_node6 = call %struct.node* @node_init()
  store %struct.node* %init_node6, %struct.node** %p
  %struct.ptr7 = getelementptr inbounds %struct.node,
%struct.node* %init_node6, i32 0, i32 1
  %malloccall8 = tail call i8* @malloc(i32 ptrtoint (i32*
getelementptr (i32, i32* null, i32 1) to i32))
  %nodeval9 = bitcast i8* %malloccall8 to i32*
  %cast10 = bitcast i32* %nodeval9 to i8*
  store i8* %cast10, i8** %struct.ptr7
  %init_node11 = call %struct.node* @node_init()
  store %struct.node* %init_node11, %struct.node** %o
  %struct.ptr12 = getelementptr inbounds %struct.node,
%struct.node* %init_node11, i32 0, i32 1
  %malloccall13 = tail call i8* @malloc(i32 ptrtoint (i32*
getelementptr (i32, i32* null, i32 1) to i32))
  %nodeval14 = bitcast i8* %malloccall13 to i32*
  %cast15 = bitcast i32* %nodeval14 to i8*
  store i8* %cast15, i8** %struct.ptr12
  %init_node16 = call %struct.node* @node_init()
  store %struct.node* %init_node16, %struct.node** %n
  %struct.ptr17 = getelementptr inbounds %struct.node,
%struct.node* %init_node16, i32 0, i32 1
  %malloccall18 = tail call i8* @malloc(i32 ptrtoint (i32*
getelementptr (i32, i32* null, i32 1) to i32))
  %nodeval19 = bitcast i8* %malloccall18 to i32*
  %cast20 = bitcast i32* %nodeval19 to i8*
  store i8* %cast20, i8** %struct.ptr17
  %init_node21 = call %struct.node* @node_init()
  store %struct.node* %init_node21, %struct.node** %m
  %struct.ptr22 = getelementptr inbounds %struct.node,
%struct.node* %init_node21, i32 0, i32 1
  %malloccall23 = tail call i8* @malloc(i32 ptrtoint (i32*
getelementptr (i32, i32* null, i32 1) to i32))
  %nodeval24 = bitcast i8* %malloccall23 to i32*
  %cast25 = bitcast i32* %nodeval24 to i8*
  store i8* %cast25, i8** %struct.ptr22
  %m26 = load %struct.node*, %struct.node** %m
  %struct.ptr27 = getelementptr inbounds %struct.node,
%struct.node* %m26, i32 0, i32 0
  store i32 1, i32* %struct.ptr27
  %n28 = load %struct.node*, %struct.node** %n
  %struct.ptr29 = getelementptr inbounds %struct.node,
%struct.node* %n28, i32 0, i32 0
  store i32 2, i32* %struct.ptr29
  %o30 = load %struct.node*, %struct.node** %o
  %struct.ptr31 = getelementptr inbounds %struct.node,
%struct.node* %o30, i32 0, i32 0
  store i32 3, i32* %struct.ptr31
  %p32 = load %struct.node*, %struct.node** %p
  %struct.ptr33 = getelementptr inbounds %struct.node,
%struct.node* %p32, i32 0, i32 0
  store i32 4, i32* %struct.ptr33
  %q34 = load %struct.node*, %struct.node** %q
  %struct.ptr35 = getelementptr inbounds %struct.node,
%struct.node* %q34, i32 0, i32 0
  store i32 5, i32* %struct.ptr35
  %r36 = load %struct.node*, %struct.node** %r
  %struct.ptr37 = getelementptr inbounds %struct.node,
%struct.node* %r36, i32 0, i32 0
  store i32 6, i32* %struct.ptr37
  %m38 = load %struct.node*, %struct.node** %m
  %n39 = load %struct.node*, %struct.node** %n
  %malloccall40 = tail call i8* @malloc(i32 ptrtoint (i32*
getelementptr (i32, i32* null, i32 1) to i32))
  %element = bitcast i8* %malloccall40 to i32*
  store i32 0, i32* %element
  %cast41 = bitcast i32* %element to i8*
  %edge_init = call %struct.edge* @edge_init(i8* %cast41,
%struct.node* %n39, i32 1)
  %edge_init42 = call %struct.edge* @edge_init(i8* %cast41,
%struct.node* %m38, i32 1)
  %struct.ptr43 = getelementptr inbounds %struct.node,
%struct.node* %m38, i32 0, i32 2
  %struct.ptr44 = getelementptr inbounds %struct.node,
%struct.node* %n39, i32 0, i32 2
  %temp = load %struct.list*, %struct.list** %struct.ptr43
  %temp45 = load %struct.list*, %struct.list** %struct.ptr44
  %cast46 = bitcast %struct.edge* %edge_init to i8*
  %cast47 = bitcast %struct.edge* %edge_init42 to i8*
  %edge_push = call i32 @list_push_back(%struct.list* %temp,
i8* %cast46)
  %edge_push48 = call i32 @list_push_back(%struct.list*
%temp45, i8* %cast47)
```

```
  %m49 = load %struct.node*, %struct.node** %m
  %m50 = load %struct.node*, %struct.node** %m
  %p51 = load %struct.node*, %struct.node** %p
  %malloccall52 = tail call i8* @malloc(i32 ptrtoint (i32*
getelementptr (i32, i32* null, i32 1) to i32))
  %element53 = bitcast i8* %malloccall52 to i32*
  store i32 0, i32* %element53
  %cast54 = bitcast i32* %element53 to i8*
  %edge_init55 = call %struct.edge* @edge_init(i8* %cast54,
%struct.node* %p51, i32 1)
  %edge_init56 = call %struct.edge* @edge_init(i8* %cast54,
%struct.node* %m50, i32 1)
  %struct.ptr57 = getelementptr inbounds %struct.node,
%struct.node* %m50, i32 0, i32 2
  %struct.ptr58 = getelementptr inbounds %struct.node,
%struct.node* %p51, i32 0, i32 2
  %temp59 = load %struct.list*, %struct.list** %struct.ptr57
  %temp60 = load %struct.list*, %struct.list** %struct.ptr58
  %cast61 = bitcast %struct.edge* %edge_init55 to i8*
  %cast62 = bitcast %struct.edge* %edge_init56 to i8*
  %edge_push63 = call i32 @list_push_back(%struct.list*
%temp59, i8* %cast61)
  %edge_push64 = call i32 @list_push_back(%struct.list*
%temp60, i8* %cast62)
  %m65 = load %struct.node*, %struct.node** %m
  %n66 = load %struct.node*, %struct.node** %n
  %o67 = load %struct.node*, %struct.node** %o
  %malloccall68 = tail call i8* @malloc(i32 ptrtoint (i32*
getelementptr (i32, i32* null, i32 1) to i32))
  %element69 = bitcast i8* %malloccall68 to i32*
  store i32 0, i32* %element69
  %cast70 = bitcast i32* %element69 to i8*
  %edge_init71 = call %struct.edge* @edge_init(i8* %cast70,
%struct.node* %o67, i32 1)
  %edge_init72 = call %struct.edge* @edge_init(i8* %cast70,
%struct.node* %n66, i32 1)
  %struct.ptr73 = getelementptr inbounds %struct.node,
%struct.node* %n66, i32 0, i32 2
  %struct.ptr74 = getelementptr inbounds %struct.node,
%struct.node* %o67, i32 0, i32 2
  %temp75 = load %struct.list*, %struct.list** %struct.ptr73
  %temp76 = load %struct.list*, %struct.list** %struct.ptr74
  %cast77 = bitcast %struct.edge* %edge_init71 to i8*
  %cast78 = bitcast %struct.edge* %edge_init72 to i8*
  %edge_push79 = call i32 @list_push_back(%struct.list*
%temp75, i8* %cast77)
  %edge_push80 = call i32 @list_push_back(%struct.list*
%temp76, i8* %cast78)
  %n81 = load %struct.node*, %struct.node** %n
  %n82 = load %struct.node*, %struct.node** %n
  %p83 = load %struct.node*, %struct.node** %p
  %malloccall84 = tail call i8* @malloc(i32 ptrtoint (i32*
getelementptr (i32, i32* null, i32 1) to i32))
  %element85 = bitcast i8* %malloccall84 to i32*
  store i32 0, i32* %element85
  %cast86 = bitcast i32* %element85 to i8*
  %edge_init87 = call %struct.edge* @edge_init(i8* %cast86,
%struct.node* %p83, i32 1)
  %edge_init88 = call %struct.edge* @edge_init(i8* %cast86,
%struct.node* %n82, i32 1)
  %struct.ptr89 = getelementptr inbounds %struct.node,
%struct.node* %n82, i32 0, i32 2
  %struct.ptr90 = getelementptr inbounds %struct.node,
%struct.node* %p83, i32 0, i32 2
  %temp91 = load %struct.list*, %struct.list** %struct.ptr89
  %temp92 = load %struct.list*, %struct.list** %struct.ptr90
  %cast93 = bitcast %struct.edge* %edge_init87 to i8*
  %cast94 = bitcast %struct.edge* %edge_init88 to i8*
  %edge_push95 = call i32 @list_push_back(%struct.list*
%temp91, i8* %cast93)
  %edge_push96 = call i32 @list_push_back(%struct.list*
%temp92, i8* %cast94)
  %n97 = load %struct.node*, %struct.node** %n
  %p98 = load %struct.node*, %struct.node** %p
  %q99 = load %struct.node*, %struct.node** %q
  %malloccall100 = tail call i8* @malloc(i32 ptrtoint (i32*
getelementptr (i32, i32* null, i32 1) to i32))
  %element101 = bitcast i8* %malloccall100 to i32*
  store i32 0, i32* %element101
  %cast102 = bitcast i32* %element101 to i8*
  %edge_init103 = call %struct.edge* @edge_init(i8* %cast102,
%struct.node* %q99, i32 1)
  %edge_init104 = call %struct.edge* @edge_init(i8* %cast102,
%struct.node* %p98, i32 1)
  %struct.ptr105 = getelementptr inbounds %struct.node,
%struct.node* %p98, i32 0, i32 2
  %struct.ptr106 = getelementptr inbounds %struct.node,
%struct.node* %q99, i32 0, i32 2
  %temp107 = load %struct.list*, %struct.list** %struct.ptr105
  %temp108 = load %struct.list*, %struct.list** %struct.ptr106
  %cast109 = bitcast %struct.edge* %edge_init103 to i8*
  %cast110 = bitcast %struct.edge* %edge_init104 to i8*
  %edge_push111 = call i32 @list_push_back(%struct.list*
%temp107, i8* %cast109)
  %edge_push112 = call i32 @list_push_back(%struct.list*
%temp108, i8* %cast110)
  %p113 = load %struct.node*, %struct.node** %p
  %q114 = load %struct.node*, %struct.node** %q
  %r115 = load %struct.node*, %struct.node** %r
  %malloccall116 = tail call i8* @malloc(i32 ptrtoint (i32*
getelementptr (i32, i32* null, i32 1) to i32))
  %element117 = bitcast i8* %malloccall116 to i32*
  store i32 0, i32* %element117
  %cast118 = bitcast i32* %element117 to i8*
  %edge_init119 = call %struct.edge* @edge_init(i8* %cast118,
%struct.node* %r115, i32 1)
  %edge_init120 = call %struct.edge* @edge_init(i8* %cast118,
%struct.node* %q114, i32 1)
  %struct.ptr121 = getelementptr inbounds %struct.node,
%struct.node* %q114, i32 0, i32 2
  %struct.ptr122 = getelementptr inbounds %struct.node,
%struct.node* %r115, i32 0, i32 2
  %temp123 = load %struct.list*, %struct.list** %struct.ptr121
  %temp124 = load %struct.list*, %struct.list** %struct.ptr122
  %cast125 = bitcast %struct.edge* %edge_init119 to i8*
  %cast126 = bitcast %struct.edge* %edge_init120 to i8*
  %edge_push127 = call i32 @list_push_back(%struct.list*
%temp123, i8* %cast125)
  %edge_push128 = call i32 @list_push_back(%struct.list*
%temp124, i8* %cast126)
  %q129 = load %struct.node*, %struct.node** %q
  %r130 = load %struct.node*, %struct.node** %r
  %m131 = load %struct.node*, %struct.node** %m
  %bfs_result = call %struct.node* @bfs(%struct.node* %m131,
%struct.node* %r130)
  ret i32 0
}

define %struct.node* @bfs(%struct.node* %root, %struct.node*
%target) {
entry:
  %root1 = alloca %struct.node*
  store %struct.node* %root, %struct.node** %root1
  %target2 = alloca %struct.node*
  store %struct.node* %target, %struct.node** %target2
  %i = alloca i32
  %origin = alloca %struct.node*
  %destination = alloca %struct.node*
  %queue = alloca %struct.list*
  %visited = alloca %struct.graph*
  %init_graph = call %struct.graph* @graph_init()
  store %struct.graph* %init_graph, %struct.graph** %visited
  %init_list = call %struct.list* @list_init()
  store %struct.list* %init_list, %struct.list** %queue
  %root3 = load %struct.node*, %struct.node** %root1
  %cast = bitcast %struct.node* %root3 to i8*
  %queue4 = load %struct.list*, %struct.list** %queue
  %list_push_back = call i32 @list_push_back(%struct.list*
%queue4, i8* %cast)
  %queue5 = load %struct.list*, %struct.list** %queue
  %init_node = call %struct.node* @node_init()
  store %struct.node* %init_node, %struct.node** %destination
  %struct.ptr = getelementptr inbounds %struct.node,
%struct.node* %init_node, i32 0, i32 1
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i32*
getelementptr (i32, i32* null, i32 1) to i32))
```

```
  %nodeval = bitcast i8* %malloccall to i32*
  %cast6 = bitcast i32* %nodeval to i8*
  store i8* %cast6, i8** %struct.ptr
  %init_node7 = call %struct.node* @node_init()
  store %struct.node* %init_node7, %struct.node** %origin
  %struct.ptr8 = getelementptr inbounds %struct.node,
%struct.node* %init_node7, i32 0, i32 1
  %malloccall9 = tail call i8* @malloc(i32 ptrtoint (i32*
getelementptr (i32, i32* null, i32 1) to i32))
  %nodeval10 = bitcast i8* %malloccall9 to i32*
  %cast11 = bitcast i32* %nodeval10 to i8*
  store i8* %cast11, i8** %struct.ptr8
  store i32 0, i32* %i
  br label %while

while:                               ; preds = %merge66, %entry
  %queue67 = load %struct.list*, %struct.list** %queue
  %struct.ptr68 = getelementptr inbounds %struct.list,
%struct.list* %queue67, i32 0, i32 0
  %struct.val.size69 = load i32, i32* %struct.ptr68
  %tmp70 = icmp sgt i32 %struct.val.size69, 0
  br i1 %tmp70, label %while_body, label %merge71

while_body:                                     ; preds = %while
  %queue12 = load %struct.list*, %struct.list** %queue
  %list_pop_front = call i8* @list_pop_front(%struct.list*
%queue12)
  %cast13 = bitcast i8* %list_pop_front to %struct.node*
  store %struct.node* %cast13, %struct.node** %origin
  %origin14 = load %struct.node*, %struct.node** %origin
  %target15 = load %struct.node*, %struct.node** %target2
  %tmp = icmp eq %struct.node* %origin14, %target15
  br i1 %tmp, label %then, label %else

merge:                                           ; preds = %else
  store i32 0, i32* %i
  br label %while25

then:                                       ; preds = %while_body
```

```
  %origin16 = load %struct.node*, %struct.node** %origin
  %struct.ptr17 = getelementptr inbounds %struct.node,
%struct.node* %origin16, i32 0, i32 0
  %struct.val.id = load i32, i32* %struct.ptr17
  %origin18 = load %struct.node*, %struct.node** %origin
  %struct.ptr19 = getelementptr inbounds %struct.node,
%struct.node* %origin18, i32 0, i32 1
  %void.ptr = load i8*, i8** %struct.ptr19
  %cast20 = bitcast i8* %void.ptr to i32*
  %struct.val.val.value = load i32, i32* %cast20
  %origin21 = load %struct.node*, %struct.node** %origin
  %struct.ptr22 = getelementptr inbounds %struct.node,
%struct.node* %origin21, i32 0, i32 2
  %struct.val.edges = load %struct.list*, %struct.list**
%struct.ptr22
  %struct.ptr23 = getelementptr inbounds %struct.list,
%struct.list* %struct.val.edges, i32 0, i32 0
  %struct.val.size = load i32, i32* %struct.ptr23
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr
inbounds ([28 x i8], [28 x i8]* @fmt.6, i32 0, i32 0), i32
%struct.val.id, i32 %struct.val.val.value, i32
%struct.val.size)
  %origin24 = load %struct.node*, %struct.node** %origin
  ret %struct.node* %origin24

else:                                      ; preds = %while_body
  br label %merge

while25:                                ; preds = %merge34, %merge
  %i59 = load i32, i32* %i
  %origin60 = load %struct.node*, %struct.node** %origin
  %struct.ptr61 = getelementptr inbounds %struct.node,
%struct.node* %origin60, i32 0, i32 2
  %struct.val.edges62 = load %struct.list*, %struct.list**
%struct.ptr61
  %struct.ptr63 = getelementptr inbounds %struct.list,
%struct.list* %struct.val.edges62, i32 0, i32 0
  %struct.val.size64 = load i32, i32* %struct.ptr63
  %tmp65 = icmp slt i32 %i59, %struct.val.size64
  br i1 %tmp65, label %while_body26, label %merge66

while_body26:                                   ; preds = %while25
  %i27 = load i32, i32* %i
  %origin28 = load %struct.node*, %struct.node** %origin
  %struct.ptr29 = getelementptr inbounds %struct.node,
%struct.node* %origin28, i32 0, i32 2
  %struct.val.edges30 = load %struct.list*, %struct.list**
%struct.ptr29
  %list_index = call i8* @list_index(%struct.list*
%struct.val.edges30, i32 %i27)
  %cast31 = bitcast i8* %list_index to %struct.edge*
  %struct.ptr32 = getelementptr inbounds %struct.edge,
%struct.edge* %cast31, i32 0, i32 2
  %struct.val.t = load i32, i32* %struct.ptr32
  %tmp33 = icmp sgt i32 %struct.val.t, 0
  br i1 %tmp33, label %then35, label %else56

merge34:                              ; preds = %else56, %merge46
  %i57 = load i32, i32* %i
  %tmp58 = add i32 %i57, 1
  store i32 %tmp58, i32* %i
  br label %while25

then35:                                    ; preds = %while_body26
  %i36 = load i32, i32* %i
  %origin37 = load %struct.node*, %struct.node** %origin
  %struct.ptr38 = getelementptr inbounds %struct.node,
%struct.node* %origin37, i32 0, i32 2
  %struct.val.edges39 = load %struct.list*, %struct.list**
%struct.ptr38
  %list_index40 = call i8* @list_index(%struct.list*
%struct.val.edges39, i32 %i36)
  %cast41 = bitcast i8* %list_index40 to %struct.edge*
  %struct.ptr42 = getelementptr inbounds %struct.edge,
%struct.edge* %cast41, i32 0, i32 1
  %struct.val.dest = load %struct.node*, %struct.node**
%struct.ptr42
  store %struct.node* %struct.val.dest, %struct.node**
```

```
%destination
  %destination43 = load %struct.node*, %struct.node**
%destination
  %visited44 = load %struct.graph*, %struct.graph** %visited
  %graph_contains_node = call i32
@graph_contains_node(%struct.graph* %visited44, %struct.node*
%destination43)
  %tmp45 = icmp eq i32 %graph_contains_node, 0
  br i1 %tmp45, label %then47, label %else55

merge46:                                ; preds = %else55, %then47
  br label %merge34

then47:                                          ; preds = %then35
  %destination48 = load %struct.node*, %struct.node**
%destination
  %visited49 = load %struct.graph*, %struct.graph** %visited
  %graph_add_node = call i32 @graph_add_node(%struct.graph*
%visited49, %struct.node* %destination48)
  %destination50 = load %struct.node*, %struct.node**
%destination
  %cast51 = bitcast %struct.node* %destination50 to i8*
  %queue52 = load %struct.list*, %struct.list** %queue
  %list_push_back53 = call i32 @list_push_back(%struct.list*
%queue52, i8* %cast51)
  %queue54 = load %struct.list*, %struct.list** %queue
  br label %merge46

else55:                                          ; preds = %then35
  br label %merge46

else56:                                    ; preds = %while_body26
  br label %merge34

merge66:                                        ; preds = %while25
  br label %while

merge71:                                          ; preds = %while
  %root72 = load %struct.node*, %struct.node** %root1
  ret %struct.node* %root72
}

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

6.2 Test Suite

Our test cases were chosen to have a near 1-1 correlation with every feature of the language, least extensively for the base microC functionality

```
/* fail-edge-mismatch.gph */
int main(){
  node<int> a;
  node<float> b;
  a <-> b;
  return 0;
}
```

```
/* fail-node-type.gph */
int main(){
    node<int> m;
    m.val = "Hello";
}
```

```
/* fail-num-args.gph */
int helper(int a, int b){
    for(a; a < b; a = a + 1){
        print(a);
    }
    return 0;
}
```

```
int main(){
    int a = 5;
    int b = 10;
    int c = 15;
    return helper(a, b, c);
}
```

```
/* fail-undecl-var.gph */
int main(){
    a = 5;
}
```

```
/* fail-wrap-type.gph */
int main(){
    graph<node<int>> g;
}
```

```
/* fail-wrong-return.gph */
void main(){
    return 0;
}
```

```
/* test-add-all.gph */
int main(){
    list<string> l;
    l.add_all("a", "b", "c", "d", "e");
    node <int> a, b, c;
    a.id = 0; a.val = 3;
    b.id = 1; b.val = 3;
    c.id = 2; c.val = 3;
    int i;
    for(i = 0; i < l.size; i = i + 1){
        print(l.peek_front());
        print(l[i]);
    }
    list<node<int>> l2;
    l2.add_all(a, b, c);
    for(i = 0; i < l2.size; i = i + 1){
        print(l2.peek_front());
        print(l2[i]);
    }
    list<float> l3;
    l3.add_all(0.1, 0.2, 0.3);
    for(i = 0; i < l3.size; i = i + 1){
        print(l3.peek_front());
        print(l3[i]);
    }
}
```

```
/* test-dedgec.gph */
int main(){
  node<string> a;
  node<string> b;
  a ->["5"] b;
  print(a.edges[0].weight);
  return 0;
}
```

```
/* test-blank-int-func.gph */
int main(){
    return 0;
}
```

```
/* test-declaration.gph */
int main(){
    int a;
    return 0;
}
```

```
/* test-dedge.gph */
int main(){
  node<float> a;
  node<float> b;
  b -> a;
  list<edge<float>> t;
  t = a.edges;
  a.val = 0.0;
  print(t[0].weight);
  print(a.val);
  return 0;
}
```

```
/* test-dedgec.gph */
int main(){
  node<string> a;
  node<string> b;
  a ->["5"] b;
  print(a.edges[0].weight);
  return 0;
}
```

```
/* test-division.gph */
int main(){
    int a = 20 / 5;
    print(a);
    return 0;
}
```

```
/* test-field.gph */
int main(){
    node<int> n, m;
    list<edge<int>> l;
    graph<int> g;
    n.val = 1;
    n.id = 0;
    m.val = 2;
    m.id = 1;
    n <->[30] m;
    g.add_node(n);
    g.add_node(m);
    list<node<int>> ll;
    ll.push_back(n);
    print(g.nodes[0].edges[0].dest.val);
    print(n.edges[0].dest.val);
    print(g.nodes[1].edges[0].dest.val);
    print(m.edges[0].dest.val);
}
```

```
/* test-function.gph */
int inc(int j){
    print(j);
    return j+1;
}

int main(){
    int j = 0;
    while(j < 10){
        j = inc(j);
    }
    return 0;
}
```

```
/* test-graph-add.gph */
int main(){
    graph<int> g;
    g.add(0, 2);
    g.add(1, 3);
    print(g[0]);
    print(g[1]);
}
```

```
/* test-graph-contains.gph */
int main(){
    graph<float> g;
    node<float> n;
    n.id = 6;
    n.val = 6.6;
    print(g.contains(n));
    g.add_node(n);
    print(g.contains(n));
    print(g.contains_id(0));
    print(g.contains_id(6));
}
```

```
/* test-graph-add.gph */
int main(){
    graph<int> g;
    g.add(0, 2);
    g.add(1, 3);
    print(g[0]);
    print(g[1]);
}
```

```
/* test-graph-decl.gph */
int main(){
    list<int> l;
    graph<int> g;
    node<int> n;
    g.add_node(n);
}
```

```
/* test-graph-get-node.gph */
int main(){
    graph<int> g;
    node<int> n, j;
    n.id = 10;
    j.id = 1;
    print(n.id);
    g.add_node(n);
    g.add_node(j);
    print(g[10].id);
    print(g[1].id);
}
```

```
/* test-if-logic.gph */
int main(){
    int a, b;
    a = 10;
    b = 20;
    if(a > b){
        print(a);
    }
    else{
        print(b);
    }
    return 0;
}
```

```
/* test-list-empty.gph */
int main(){
    list<int> l;
    print(l.size == 0);
    l.push_back(10);
    print(l.size == 0);
}
```

```
/* test-list-float.gph */
int main(){
    list<float> i, j;
    i.push_back(1.1);
    j.push_back(2.2);
    print(i[0]);
    print(j[0]);
}
```

```
/* test-list-index.gph */
int main(){
    list<int> l, l1;
    l.push_back(3);
    l.push_back(5);
    print(l[0]);
    print(l[1]);
    print(l.size);
    print(l1.size);
    return 0;
}
```

```
/* test-list-peek.gph */
int main(){
    list<string> l;
    l.push_back("a");
    l.push_back("b");
    print(l.size);
    print(l.peek_front());
    print(l.peek_back());
    print(l.size);
}
```

```
/* test-list-pop-back.gph */
int main(){
    list<int> l;
    l.push_back(1);
    l.push_back(2);
    print(l.pop_back());
    print(l.pop_back());
}
```

```
/* test-list-pop-front.gph */
int main(){
    list<int> l;
    l.push_back(1);
    l.push_back(2);
    print(l.pop_front());
    print(l.pop_front());
}
```

```
/* test-list-push-back.gph */
int main(){
    list<int> l;
    l.push_back(1);
    l.push_back(2);
    print(l[0]);
    print(l[1]);
}
```

```
/* test-list-push-front.gph */
int main(){
    list<int> l;
    l.push_front(1);
    l.push_front(2);
    print(l[0]);
    print(l[1]);
    return 0;
}
```

```
/* test-multiplication.gph */
int main(){
    int a = 5 * 20;
    print(a);
    return 0;
}
```

```
/* test-node-edges.gph */
int main(){
    node<int> n, o, p, q;
    o.id = 0; p.id = 1; q.id = 2;
    q -> o -> p;
    n.val = 20;
    n.id = 30;
    list<int> l;
    l.push_back(1);
    print(l[0]);
    list<edge<int>> j = q.edges;
    print(q.edges.size);
    return 0;
}
```

```
/* test-node-float.gph */
int main(){
  node<float> j;
  j.val = 0.1;
  print(j.val);
  return 0;
}
```

```
/* test-node-id.gph */
int main(){
  node<int> j;
  node<float> k;
  j.id = 9;
  k.id = 0;
  print(j.id);
  print(k.id);
  return 0;
}
```

```
/* test-node-int.gph */
int main(){
  node<int> j;
  j.val = 1;
  print(j.val);
  return 0;
}
```

```
/* test-node.gph */
int main() {
  node <int> j;
  j.id = 8;
  print(j.id);
  return 0;
}
```

```
/* test-pointer-comp.gph */
int main(){
    node<int> a, b;
    cmp(a, b);
    a = b;
    cmp(a, b);
}

void cmp(node<int> a, node<int> b){
    if(a == b){
        print("yes");
    }
    else{
        print("no");
    }
}
```

```
/* test-string.gph */
int main(){
    list<string> l;
    l.push_back("1");
    l.push_back("2");
    l.push_back("3");
    while(l.size > 0) {
        print(l.pop_back());
    }
    string s = "hello";
    print(s);
    node<string> n;
    n.val = "world";
    print(n.val);
}
```

```
/* test-subtraction.gph */
int main(){
    int a= 5 - 20;
    print(a);
    return 0;
}
```

```
/* test-uedge.gph */
int main(){
  node <int> a;
  node <int> b;
  a <-> b;
  print(b.edges[0].weight);
  return 0;
}
```

```
/* test-uedgec.gph */
int main(){
  node<int> a;
  node<int> b;
  a <->[3] b;
  return 0;
}
```

```
/* test-uprint.gph */
int main(){
    int a = 1;
    float b = 2.0;
    string c = "3";
    node<string> n, o, p, q, r, s, t;
    n.id = 4;
    n.val = "5";
    n -> o;
    n -> p;
    n -> q;
    n -> r;
    n -> s;
    n -> t;
    print(a);
    print(b);
    print(c);
    print(n);
}
```

```
/* test-vardecl.gph */
int main(){
  int a;
  a = 2;
  int b = 3;
  int c = a + b;
  print(c);
  return 0;
}
```

6.3 Test Automation

All testing was automated for the Graphene programming language. A bash script, **testall.sh**, executed every integration test withinour test suite and matched the results with the expected results. For each test, it was clearly denoted whether or not the test passed or failed, and for failed test cases, the script output the actual error that occured. Because of the ease of running the test script, team members constantly ran the integration tests whenever changes were made to the code to ensure at each and every step we were on the right track. This falls in line with test driven development. Below is the output from the end to end integration test.

```
/usr/bin/lli

###### Testing test-add-all
./graphene.native tests/test-add-all.gph > test-add-all.ll
llc -relocation-model=pic test-add-all.ll > test-add-all.s
cc -o test-add-all.exe test-add-all.s graphene.o
./test-add-all.exe
diff -b test-add-all.out tests/test-add-all.out > test-add-all.diff
###### SUCCESS

###### Testing test-addition
./graphene.native tests/test-addition.gph > test-addition.ll
llc -relocation-model=pic test-addition.ll > test-addition.s
cc -o test-addition.exe test-addition.s graphene.o
./test-addition.exe
diff -b test-addition.out tests/test-addition.out >
test-addition.diff
###### SUCCESS

###### Testing test-blank-int-func
./graphene.native tests/test-blank-int-func.gph >
test-blank-int-func.ll
llc -relocation-model=pic test-blank-int-func.ll >
test-blank-int-func.s
cc -o test-blank-int-func.exe test-blank-int-func.s graphene.o
./test-blank-int-func.exe
diff -b test-blank-int-func.out tests/test-blank-int-func.out >
test-blank-int-func.diff
###### SUCCESS

###### Testing test-declaration
./graphene.native tests/test-declaration.gph > test-declaration.ll
llc -relocation-model=pic test-declaration.ll > test-declaration.s
cc -o test-declaration.exe test-declaration.s graphene.o
./test-declaration.exe
diff -b test-declaration.out tests/test-declaration.out >
test-declaration.diff
###### SUCCESS

###### Testing test-dedge
./graphene.native tests/test-dedge.gph > test-dedge.ll
llc -relocation-model=pic test-dedge.ll > test-dedge.s
cc -o test-dedge.exe test-dedge.s graphene.o
```

```
./test-dedge.exe
diff -b test-dedge.out tests/test-dedge.out > test-dedge.diff
###### SUCCESS

###### Testing test-dedgec
./graphene.native tests/test-dedgec.gph > test-dedgec.ll
llc -relocation-model=pic test-dedgec.ll > test-dedgec.s
cc -o test-dedgec.exe test-dedgec.s graphene.o
./test-dedgec.exe
diff -b test-dedgec.out tests/test-dedgec.out > test-dedgec.diff
###### SUCCESS

###### Testing test-demo-sr-parse
./graphene.native tests/test-demo-sr-parse.gph >
test-demo-sr-parse.ll
llc -relocation-model=pic test-demo-sr-parse.ll >
test-demo-sr-parse.s
cc -o test-demo-sr-parse.exe test-demo-sr-parse.s graphene.o
./test-demo-sr-parse.exe
diff -b test-demo-sr-parse.out tests/test-demo-sr-parse.out >
test-demo-sr-parse.diff
###### SUCCESS

###### Testing test-division
./graphene.native tests/test-division.gph > test-division.ll
llc -relocation-model=pic test-division.ll > test-division.s
cc -o test-division.exe test-division.s graphene.o
./test-division.exe
diff -b test-division.out tests/test-division.out >
test-division.diff
###### SUCCESS

###### Testing test-field
./graphene.native tests/test-field.gph > test-field.ll
llc -relocation-model=pic test-field.ll > test-field.s
cc -o test-field.exe test-field.s graphene.o
./test-field.exe
diff -b test-field.out tests/test-field.out > test-field.diff
###### SUCCESS

###### Testing test-function
./graphene.native tests/test-function.gph > test-function.ll
llc -relocation-model=pic test-function.ll > test-function.s
```

```
cc -o test-function.exe test-function.s graphene.o
./test-function.exe
diff -b test-function.out tests/test-function.out >
test-function.diff
###### SUCCESS

###### Testing test-graph-add
./graphene.native tests/test-graph-add.gph > test-graph-add.ll
llc -relocation-model=pic test-graph-add.ll > test-graph-add.s
cc -o test-graph-add.exe test-graph-add.s graphene.o
./test-graph-add.exe
diff -b test-graph-add.out tests/test-graph-add.out >
test-graph-add.diff
###### SUCCESS

###### Testing test-graph-contains
./graphene.native tests/test-graph-contains.gph >
test-graph-contains.ll
llc -relocation-model=pic test-graph-contains.ll >
test-graph-contains.s
cc -o test-graph-contains.exe test-graph-contains.s graphene.o
./test-graph-contains.exe
diff -b test-graph-contains.out tests/test-graph-contains.out >
test-graph-contains.diff
###### SUCCESS

###### Testing test-graph-decl
./graphene.native tests/test-graph-decl.gph > test-graph-decl.ll
llc -relocation-model=pic test-graph-decl.ll > test-graph-decl.s
cc -o test-graph-decl.exe test-graph-decl.s graphene.o
./test-graph-decl.exe
diff -b test-graph-decl.out tests/test-graph-decl.out >
test-graph-decl.diff
###### SUCCESS

###### Testing test-graph-get-node
./graphene.native tests/test-graph-get-node.gph >
test-graph-get-node.ll
llc -relocation-model=pic test-graph-get-node.ll >
test-graph-get-node.s
cc -o test-graph-get-node.exe test-graph-get-node.s graphene.o
./test-graph-get-node.exe
diff -b test-graph-get-node.out tests/test-graph-get-node.out >
```

```
test-graph-get-node.diff
###### SUCCESS

###### Testing test-if-logic
./graphene.native tests/test-if-logic.gph > test-if-logic.ll
llc -relocation-model=pic test-if-logic.ll > test-if-logic.s
cc -o test-if-logic.exe test-if-logic.s graphene.o
./test-if-logic.exe
diff -b test-if-logic.out tests/test-if-logic.out >
test-if-logic.diff
###### SUCCESS

###### Testing test-list-empty
./graphene.native tests/test-list-empty.gph > test-list-empty.ll
llc -relocation-model=pic test-list-empty.ll > test-list-empty.s
cc -o test-list-empty.exe test-list-empty.s graphene.o
./test-list-empty.exe
diff -b test-list-empty.out tests/test-list-empty.out >
test-list-empty.diff
###### SUCCESS

###### Testing test-list-float
./graphene.native tests/test-list-float.gph > test-list-float.ll
llc -relocation-model=pic test-list-float.ll > test-list-float.s
cc -o test-list-float.exe test-list-float.s graphene.o
./test-list-float.exe
diff -b test-list-float.out tests/test-list-float.out >
test-list-float.diff
###### SUCCESS

###### Testing test-list-index
./graphene.native tests/test-list-index.gph > test-list-index.ll
llc -relocation-model=pic test-list-index.ll > test-list-index.s
cc -o test-list-index.exe test-list-index.s graphene.o
./test-list-index.exe
diff -b test-list-index.out tests/test-list-index.out >
test-list-index.diff
###### SUCCESS

###### Testing test-list-peek
./graphene.native tests/test-list-peek.gph > test-list-peek.ll
llc -relocation-model=pic test-list-peek.ll > test-list-peek.s
cc -o test-list-peek.exe test-list-peek.s graphene.o
```

```
./test-list-peek.exe
diff -b test-list-peek.out tests/test-list-peek.out >
test-list-peek.diff
###### SUCCESS

###### Testing test-list-pop-back
./graphene.native tests/test-list-pop-back.gph >
test-list-pop-back.ll
llc -relocation-model=pic test-list-pop-back.ll >
test-list-pop-back.s
cc -o test-list-pop-back.exe test-list-pop-back.s graphene.o
./test-list-pop-back.exe
diff -b test-list-pop-back.out tests/test-list-pop-back.out >
test-list-pop-back.diff
###### SUCCESS

###### Testing test-list-pop-front
./graphene.native tests/test-list-pop-front.gph >
test-list-pop-front.ll
llc -relocation-model=pic test-list-pop-front.ll >
test-list-pop-front.s
cc -o test-list-pop-front.exe test-list-pop-front.s graphene.o
./test-list-pop-front.exe
diff -b test-list-pop-front.out tests/test-list-pop-front.out >
test-list-pop-front.diff
###### SUCCESS

###### Testing test-list-push-back
./graphene.native tests/test-list-push-back.gph >
test-list-push-back.ll
llc -relocation-model=pic test-list-push-back.ll >
test-list-push-back.s
cc -o test-list-push-back.exe test-list-push-back.s graphene.o
./test-list-push-back.exe
diff -b test-list-push-back.out tests/test-list-push-back.out >
test-list-push-back.diff
###### SUCCESS

###### Testing test-list-push-front
./graphene.native tests/test-list-push-front.gph >
test-list-push-front.ll
llc -relocation-model=pic test-list-push-front.ll >
test-list-push-front.s
```

```
cc -o test-list-push-front.exe test-list-push-front.s graphene.o
./test-list-push-front.exe
diff -b test-list-push-front.out tests/test-list-push-front.out >
test-list-push-front.diff
###### SUCCESS

###### Testing test-multiplication
./graphene.native tests/test-multiplication.gph >
test-multiplication.ll
llc -relocation-model=pic test-multiplication.ll >
test-multiplication.s
cc -o test-multiplication.exe test-multiplication.s graphene.o
./test-multiplication.exe
diff -b test-multiplication.out tests/test-multiplication.out >
test-multiplication.diff
###### SUCCESS

###### Testing test-node-edges
./graphene.native tests/test-node-edges.gph > test-node-edges.ll
llc -relocation-model=pic test-node-edges.ll > test-node-edges.s
cc -o test-node-edges.exe test-node-edges.s graphene.o
./test-node-edges.exe
diff -b test-node-edges.out tests/test-node-edges.out >
test-node-edges.diff
###### SUCCESS

###### Testing test-node-float
./graphene.native tests/test-node-float.gph > test-node-float.ll
llc -relocation-model=pic test-node-float.ll > test-node-float.s
cc -o test-node-float.exe test-node-float.s graphene.o
./test-node-float.exe
diff -b test-node-float.out tests/test-node-float.out >
test-node-float.diff
###### SUCCESS

###### Testing test-node-id
./graphene.native tests/test-node-id.gph > test-node-id.ll
llc -relocation-model=pic test-node-id.ll > test-node-id.s
cc -o test-node-id.exe test-node-id.s graphene.o
./test-node-id.exe
diff -b test-node-id.out tests/test-node-id.out > test-node-id.diff
###### SUCCESS
```

```
###### Testing test-node-int
./graphene.native tests/test-node-int.gph > test-node-int.ll
llc -relocation-model=pic test-node-int.ll > test-node-int.s
cc -o test-node-int.exe test-node-int.s graphene.o
./test-node-int.exe
diff -b test-node-int.out tests/test-node-int.out >
test-node-int.diff
###### SUCCESS

###### Testing test-node
./graphene.native tests/test-node.gph > test-node.ll
llc -relocation-model=pic test-node.ll > test-node.s
cc -o test-node.exe test-node.s graphene.o
./test-node.exe
diff -b test-node.out tests/test-node.out > test-node.diff
###### SUCCESS

###### Testing test-pointer-comp
./graphene.native tests/test-pointer-comp.gph > test-pointer-comp.ll
llc -relocation-model=pic test-pointer-comp.ll > test-pointer-comp.s
cc -o test-pointer-comp.exe test-pointer-comp.s graphene.o
./test-pointer-comp.exe
diff -b test-pointer-comp.out tests/test-pointer-comp.out >
test-pointer-comp.diff
###### SUCCESS

###### Testing test-report-bfs
./graphene.native tests/test-report-bfs.gph > test-report-bfs.ll
llc -relocation-model=pic test-report-bfs.ll > test-report-bfs.s
cc -o test-report-bfs.exe test-report-bfs.s graphene.o
./test-report-bfs.exe
diff -b test-report-bfs.out tests/test-report-bfs.out >
test-report-bfs.diff
###### SUCCESS

###### Testing test-report-edges
./graphene.native tests/test-report-edges.gph > test-report-edges.ll
llc -relocation-model=pic test-report-edges.ll > test-report-edges.s
cc -o test-report-edges.exe test-report-edges.s graphene.o
./test-report-edges.exe
diff -b test-report-edges.out tests/test-report-edges.out >
test-report-edges.diff
###### SUCCESS
```

```
###### Testing test-report-functions
./graphene.native tests/test-report-functions.gph >
test-report-functions.ll
llc -relocation-model=pic test-report-functions.ll >
test-report-functions.s
cc -o test-report-functions.exe test-report-functions.s graphene.o
./test-report-functions.exe
diff -b test-report-functions.out tests/test-report-functions.out >
test-report-functions.diff
###### SUCCESS

###### Testing test-report-graphs
./graphene.native tests/test-report-graphs.gph >
test-report-graphs.ll
llc -relocation-model=pic test-report-graphs.ll >
test-report-graphs.s
cc -o test-report-graphs.exe test-report-graphs.s graphene.o
./test-report-graphs.exe
diff -b test-report-graphs.out tests/test-report-graphs.out >
test-report-graphs.diff
###### SUCCESS

###### Testing test-report-lists
./graphene.native tests/test-report-lists.gph > test-report-lists.ll
llc -relocation-model=pic test-report-lists.ll > test-report-lists.s
cc -o test-report-lists.exe test-report-lists.s graphene.o
./test-report-lists.exe
diff -b test-report-lists.out tests/test-report-lists.out >
test-report-lists.diff
###### SUCCESS

###### Testing test-report-logic
./graphene.native tests/test-report-logic.gph > test-report-logic.ll
llc -relocation-model=pic test-report-logic.ll > test-report-logic.s
cc -o test-report-logic.exe test-report-logic.s graphene.o
./test-report-logic.exe
diff -b test-report-logic.out tests/test-report-logic.out >
test-report-logic.diff
###### SUCCESS

###### Testing test-report-loops
./graphene.native tests/test-report-loops.gph > test-report-loops.ll
```

```
llc -relocation-model=pic test-report-loops.ll > test-report-loops.s
cc -o test-report-loops.exe test-report-loops.s graphene.o
./test-report-loops.exe
diff -b test-report-loops.out tests/test-report-loops.out >
test-report-loops.diff
###### SUCCESS

###### Testing test-report-main
./graphene.native tests/test-report-main.gph > test-report-main.ll
llc -relocation-model=pic test-report-main.ll > test-report-main.s
cc -o test-report-main.exe test-report-main.s graphene.o
./test-report-main.exe
diff -b test-report-main.out tests/test-report-main.out >
test-report-main.diff
###### SUCCESS

###### Testing test-report-nodes
./graphene.native tests/test-report-nodes.gph > test-report-nodes.ll
llc -relocation-model=pic test-report-nodes.ll > test-report-nodes.s
cc -o test-report-nodes.exe test-report-nodes.s graphene.o
./test-report-nodes.exe
diff -b test-report-nodes.out tests/test-report-nodes.out >
test-report-nodes.diff
###### SUCCESS

###### Testing test-report-primitive
./graphene.native tests/test-report-primitive.gph >
test-report-primitive.ll
llc -relocation-model=pic test-report-primitive.ll >
test-report-primitive.s
cc -o test-report-primitive.exe test-report-primitive.s graphene.o
./test-report-primitive.exe
diff -b test-report-primitive.out tests/test-report-primitive.out >
test-report-primitive.diff
###### SUCCESS

###### Testing test-string-cmp
./graphene.native tests/test-string-cmp.gph > test-string-cmp.ll
llc -relocation-model=pic test-string-cmp.ll > test-string-cmp.s
cc -o test-string-cmp.exe test-string-cmp.s graphene.o
./test-string-cmp.exe
diff -b test-string-cmp.out tests/test-string-cmp.out >
test-string-cmp.diff
```

```
###### SUCCESS

###### Testing test-string
./graphene.native tests/test-string.gph > test-string.ll
llc -relocation-model=pic test-string.ll > test-string.s
cc -o test-string.exe test-string.s graphene.o
./test-string.exe
diff -b test-string.out tests/test-string.out > test-string.diff
###### SUCCESS

###### Testing test-subtraction
./graphene.native tests/test-subtraction.gph > test-subtraction.ll
llc -relocation-model=pic test-subtraction.ll > test-subtraction.s
cc -o test-subtraction.exe test-subtraction.s graphene.o
./test-subtraction.exe
diff -b test-subtraction.out tests/test-subtraction.out >
test-subtraction.diff
###### SUCCESS

###### Testing test-uedge
./graphene.native tests/test-uedge.gph > test-uedge.ll
llc -relocation-model=pic test-uedge.ll > test-uedge.s
cc -o test-uedge.exe test-uedge.s graphene.o
./test-uedge.exe
diff -b test-uedge.out tests/test-uedge.out > test-uedge.diff
###### SUCCESS

###### Testing test-uedgec
./graphene.native tests/test-uedgec.gph > test-uedgec.ll
llc -relocation-model=pic test-uedgec.ll > test-uedgec.s
cc -o test-uedgec.exe test-uedgec.s graphene.o
./test-uedgec.exe
diff -b test-uedgec.out tests/test-uedgec.out > test-uedgec.diff
###### SUCCESS

###### Testing test-uprint
./graphene.native tests/test-uprint.gph > test-uprint.ll
llc -relocation-model=pic test-uprint.ll > test-uprint.s
cc -o test-uprint.exe test-uprint.s graphene.o
./test-uprint.exe
diff -b test-uprint.out tests/test-uprint.out > test-uprint.diff
###### SUCCESS
```

```
###### Testing test-vardecl
./graphene.native tests/test-vardecl.gph > test-vardecl.ll
llc -relocation-model=pic test-vardecl.ll > test-vardecl.s
cc -o test-vardecl.exe test-vardecl.s graphene.o
./test-vardecl.exe
diff -b test-vardecl.out tests/test-vardecl.out > test-vardecl.diff
###### SUCCESS

###### Testing fail-edge-mismatch
./graphene.native < tests/fail-edge-mismatch.gph 2>
fail-edge-mismatch.err >> testall.log
diff -b fail-edge-mismatch.err tests/fail-edge-mismatch.err >
fail-edge-mismatch.diff
###### SUCCESS

###### Testing fail-node-type
./graphene.native < tests/fail-node-type.gph 2> fail-node-type.err >>
testall.log
diff -b fail-node-type.err tests/fail-node-type.err >
fail-node-type.diff
###### SUCCESS

###### Testing fail-num-args
./graphene.native < tests/fail-num-args.gph 2> fail-num-args.err >>
testall.log
diff -b fail-num-args.err tests/fail-num-args.err >
fail-num-args.diff
###### SUCCESS

###### Testing fail-undecl-var
./graphene.native < tests/fail-undecl-var.gph 2> fail-undecl-var.err
>> testall.log
diff -b fail-undecl-var.err tests/fail-undecl-var.err >
fail-undecl-var.diff
###### SUCCESS

###### Testing fail-wrap-type
./graphene.native < tests/fail-wrap-type.gph 2> fail-wrap-type.err >>
testall.log
diff -b fail-wrap-type.err tests/fail-wrap-type.err >
fail-wrap-type.diff
###### SUCCESS
```

```
###### Testing fail-wrong-return
./graphene.native < tests/fail-wrong-return.gph 2>
fail-wrong-return.err >> testall.log
diff -b fail-wrong-return.err tests/fail-wrong-return.err >
fail-wrong-return.diff
###### SUCCESS
```

7. Lessons Learned

7.1 Ashar Nadeem

Over the course of this project, lots of lessons were learned, knowledge was gained, and tears were shed. Working in a functional language for the first time, I would say understanding how to do basic programming in OCaml was among the more challenging parts initially. Similarly, trying to understand what was happening in the codebases for past projects as well as MicroC was a grueling, but very rewarding process. Getting the chance to build a language from scratch, from the barebones scanner all the way up to the semantic checker and code generation was very rewarding, and seeing it all come together and being able to code in my own custom language was an amazing experience.

Getting started early on this project is essential. Because of the aforementioned learning curve, I wish we would have started working on the codegen even earlier as we would have been able to implement so much more functionality by the time we learned how to more efficiently bring our ideas to life. Additionally, sticking to a strict timeline also helps to bring things together. Having to wait on other team members to complete some dependency, or procrastinating until the last day led to some inefficiencies/shortcomings in our project. These, while not critical, still lead to a less than satisfactory result from a perfectionist point of view.

Lastly, as with all group projects, choosing your teammates very wisely and holding them accountable cannot be stressed enough. Our group was one of the few train wrecks during the semester, as certain group members had to be kicked due to inactivity while others got by without contributing a single line of code to the project. Looking back, I would have liked to set some crystal clear expectations for everyone as well as myself, and addressed issues earlier in the semester rather than later.

7.2 Matthew Sanchez

This project had a lot of "firsts" for me. This was my first foray into functional programming, my first time working with such a low-level API, my first time working on a major programming project in a group. In regards to functional programming, I absolutely love it. I've never been as excited about learning a new language as I was about learning OCaml, and I'm glad this project was extensive enough to force students to go very deep into it and demonstrate a case where functional languages thrive. In regards to LLVM, it was a great learning experience. It was very intimidating at first, especially when I was new to OCaml, but once I learned to access the many resources made available to students (microC and past projects) and took the time to understand every single line of code in such resources, LLVM no longer looked like hellish gibberish. My systems fundamentals (CSEE 3827) background was not the strongest coming into this course, and the task of working with and understanding how to work with LLVM was a perfect opportunity to fix that. This process also provided a ton of insight into how compilers for other languages might work. In regards to working with a team, it was an archetypal Greek tragedy. I was most worried about this part of the class upon registration, and this turned out to be what went most poorly. Group communication was poor to start this project, partly due to the lack of physical proximity in pandemic learning. This initial lack of communication led to a massively disproportionate workload in the group. Some members would anticipate the increasing amounts of work to meet each deadline and take it upon themselves to make ends meet while more passive members of the group allowed work to be done by others with a promise of contributing "next time". This snowballed to the point where we had to kick someone out of our group because familiarizing them with the code-base would have been more work than simply finishing it without the extra hands. The group aspect can be the single point of failure for this otherwise fantastic class, and you should prepare for the worst so the worst doesn't happen.

7.3 Vasileios Kopanas

During the duration of the semester, it was really fun learning a new programming language, that I actually never knew it existed. Also, throughout the semester, I believe this project forced me to try to understand how LLVM works (which is far from easy), by going through all the resources offered. This project highlighted the importance of team work for a successful and well-running project. Especially when so many agents were involved in this single project where the work of each one of us could crucially affect other parts, which my teammate's have already come up with, or were in the process of completing made me realize the importance of scheduling meetings and communicating in a cohesive way. Making a coherent plan and dividing clearly what needs to be carried out by each teammate would be my advice to prospective students, in order to save time and for each teammate to be particularly focused on their own part and achieve a more detailed and better result. Furthermore, I am also really keen in asking whatever you don't know and also studying the other parts of the project: not all members contributed to the coding from the beginning which made it difficult to understand how the project was progressing and it also made it difficult add to it on the later stages.

8. Appendix

8.1 scanner.mll

```
(*
    Ocamlyacc scanner for Graphene
    Author: Ashar Nadeem and Matthew Sanchez
*)

{ open Parser }

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

rule token = parse
 | eof                     { EOF }
(* ignored *)
 | [' ' '\t' '\r' '\n']  { token lexbuf }
 | "//"                    { linecomment lexbuf }parser | "/*"
{ longcomment lexbuf }
(* blocking *)
 | '('                     { LPAREN }
 | ')'                     { RPAREN }
 | '{'                     { LBRACE }
 | '}'                     { RBRACE }
 | '['                     { LSQUARE }
 | ']'                     { RSQUARE }
 | ';'                     { SEMI }
 | ':'                     { COLON }
 | ','                     { COMMA }
(* operators *)
 | '+'                     { PLUS }
 | '-'                     { MINUS }
 | '*'                     { TIMES }
 | '/'                     { DIVIDE }
 | '='                     { ASSIGN }
 | '%'                     { MOD }
 | "=="                    { EQ }
 | '.'                     { DOT }
 | '<'                     { LT }
 | '>'                     { GT }
 | "<="                    { LEQ }
 | ">="                    { GEQ }
 | "!="                    { NEQ }
 | "&&"                    { AND }
 | "||"                    { OR }
 | "!"                     { NOT }
(* graph tokens *)
 | "->"            { DEDGE }
 | "<->"           { UEDGE }

(* keywords *)
 | "void"                  { VOID }
 | "int"                   { INT }
 | "float"                 { FLOAT }
 | "string"                { STRING }
 | "graph"                 { GRAPH }
 | "node"                  { NODE }
 | "edge"                  { EDGE }
 | "list"                  { LIST }
 | "if"                    { IF }
```

```
  | "else"                 { ELSE }
  | "for"                  { FOR }
  | "while"                { WHILE }
  | "return"               { RETURN }
  | "print"                { PRINT }
  | "push_back"            { PUSHBACK }
  | "push_front"           { PUSHFRONT }
  | "pop_back"             { POPBACK }
  | "pop_front"            { POPFRONT }
  | "peek_back"            { PEEKBACK }
  | "peek_front"           { PEEKFRONT }
  | "add_all"              { ADDALL }
  | "contains"             { CONTAINS }
  | "contains_id"          { CONTAINSID }
  | "add_node"             { ADDNODE }
  | "add"                  { ADD }

(* literals *)
  | '\"' ([^'\"']* as str) '\"' { SLIT(str) }
  | digits as num { LITERAL(int_of_string num) }
  | (('0'|['1'-'9']['0'-'9']*) '.' ['0'-'9']+) as num { FLIT(num) }
  | ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as str { ID(str) }

 and linecomment = parse
    '\n'                   { token lexbuf }
  | _                      { linecomment lexbuf }

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

8.2 parser.mly

```
/*
    Ocamlyacc parser for Graphene
    Author: Matthew Sanchez
*/

%{ open Ast %}
```

```
%token LPAREN RPAREN LBRACE RBRACE LSQUARE RSQUARE COMMA SEMI
COLON DOT
%token PLUS MINUS TIMES DIVIDE MOD ASSIGN NOT EQ NEQ LT LEQ GT
GEQ AND OR
%token DEDGE UEDGE
%token RETURN IF ELSE FOR WHILE PRINT
%token INT FLOAT STRING GRAPH NODE EDGE LIST VOID
%token PUSHBACK PUSHFRONT POPBACK POPFRONT PEEKBACK PEEKFRONT
%token CONTAINS CONTAINSID ADDALL ADDNODE ADD
%token <int> LITERAL
%token <string> FLIT
%token <string> SLIT
%token <string> ID
%token EOF

%start program
%type <Ast.program> program

%nonassoc NOELSE
%nonassoc ELSE
%right DIREDGE UNDIREDGE TILDE DGT DEDGE UEDGE DEDGEP UEDGE
%right ASSIGN LSQUARE RSQUARE
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE MOD
%right NOT
%right DOT
%%

program:
 decls EOF { $1 }

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

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

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

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

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

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_opt SEMI expr_opt RPAREN stmt
                                                       {
For($3, $5, $7, $9) }
 | WHILE LPAREN expr RPAREN stmt { While($3, $5) }
 | typ ID ASSIGN expr SEMI { Declare($1, [$2], Assign($2, $4))
}
 | typ id_list SEMI { Declare($1, $2, Noexpr) }

id_list:
   ID                { [ $1 ] }
 | id_list COMMA ID  { $3 :: $1 }

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

expr:
   literal { $1 }
 | expr PLUS expr { Binop($1, Add, $3) }
 | expr MINUS expr { Binop($1, Sub, $3) }
 | expr TIMES expr { Binop($1, Mul, $3) }
 | expr DIVIDE expr { Binop($1, Div, $3) }
 | expr MOD expr { Binop($1, Mod, $3) }
 | expr EQ expr { Binop($1, Eq, $3) }
 | expr NEQ expr { Binop($1, Neq, $3) }
 | expr LT expr { Binop($1, Less, $3) }
 | expr GT expr { Binop($1, Greater, $3) }
 | expr LEQ expr { Binop($1, Leq, $3) }
 | expr GEQ expr { Binop($1, Geq, $3) }
 | expr AND expr { Binop($1, And, $3) }
 | expr OR expr { Binop($1, Or, $3) }
 | MINUS expr %prec NOT { Unop(Neg, $2) }
 | NOT expr { Unop(Not, $2) }
 | ID ASSIGN expr { Assign($1, $3) }
 | expr DOT ID ASSIGN expr { AssignField($1, $3, $5) }
 | LPAREN expr RPAREN { $2 }
 /* nodeA -> nodeB */
 | expr DEDGE expr { DEdge( $1, $3) }
 /* nodeA ->[1] nodeB */
 | expr DEDGE LSQUARE expr RSQUARE expr { DEdgeC( $1, $4, $6) }
 /* nodeA <-> nodeB */
 | expr UEDGE expr { UEdge ( $1, $3) }
 /* nodeA <-> nodeB */
 | expr UEDGE LSQUARE expr RSQUARE expr { UEdgeC( $1, $4, $6) }
 /* node.id */
 | expr DOT ID { Access($1, $3) }
 /* list[3] */
 | expr LSQUARE expr RSQUARE { Index($1, $3) }
 | ID LPAREN args_opt RPAREN { Call($1, $3) }
 | PRINT LPAREN expr RPAREN { Print($3) }
 | expr DOT ID LPAREN args_opt RPAREN { Call( $3, $1 :: $5 ) }
 | expr DOT PUSHBACK LPAREN expr RPAREN { PushBack($1, $5) }
 | expr DOT PUSHFRONT LPAREN expr RPAREN { PushFront($1, $5) }
 | expr DOT POPBACK LPAREN RPAREN { PopBack($1) }
 | expr DOT POPFRONT LPAREN RPAREN { PopFront($1) }
 | expr DOT PEEKBACK LPAREN RPAREN { PeekBack($1) }
 | expr DOT PEEKFRONT LPAREN RPAREN { PeekFront($1) }
 | expr DOT ADDNODE LPAREN expr RPAREN { AddNode($1, $5) }
 | expr DOT ADD LPAREN expr COMMA expr RPAREN { GAdd($1, $5,
$7) }
 | expr DOT CONTAINS LPAREN expr RPAREN { Contains($1, $5) }
 | expr DOT CONTAINSID LPAREN expr RPAREN { ContainsId($1, $5)
}
 | expr DOT ADDALL LPAREN args_list RPAREN { AddAll( $1,
List.rev $5 )}

literal:
   ID { Id($1) }
 | LITERAL { Ilit($1) }
 | FLIT { Flit($1) }
 | SLIT { Slit($1) }

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

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

8.3 ast.ml

```
(*
     Abstract Syntax Tree
     Author: Matthew Sanchez
*)

type unop =
   Not
 | Neg

type binop =
   Add
 | Sub
 | Mul
 | Div
 | Mod
 | Eq
 | And
 | Or
 | Greater
 | Less
 | Geq
 | Leq
 | Neq

type typ =
   Int
 | String
 | Float
 | Void
 | Graph of typ
 | Edge of typ
 | Node of typ
 | List of typ
```

```
type expr =
   Ilit of int
 | Slit of string
 | Flit of string
 | Id of string
 | Unop of unop * expr
 | Binop of expr * binop * expr
 | Assign of string * expr
 | AssignField of expr * string * expr
 | Call of string * expr list
 | Print of expr
 | Access of expr * string
 | Index of expr * expr
 | PushBack of expr * expr
 | PushFront of expr * expr
 | PopBack of expr
 | PopFront of expr
 | PeekBack of expr
 | PeekFront of expr
 | AddNode of expr * expr
 | Noexpr
 | UEdge of expr * expr
 | UEdgeC of expr * expr * expr
 | DEdge of expr * expr
 | DEdgeC of expr * expr * expr
 | GAdd of expr * expr * expr
 | Contains of expr * expr
 | ContainsId of expr * expr
 | AddAll of expr * expr list

type bind = typ * string

type stmt =
   Expr of expr
 | Block of stmt list
 | Return of expr
 | If of expr * stmt * stmt
 | For of expr * expr * expr * stmt
 | While of expr * stmt
 | Declare of typ * string list * expr

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

type program = bind list * func_decl list

(* Pretty-printing functions *)

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

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


let rec string_of_typ = function
   Int -> "int"
 | String -> "string"
 | Float -> "float"
 | Void -> "void"
 | Graph(t) -> "graph<" ^ string_of_typ t ^ ">"
 | Node(t) -> "node<" ^ string_of_typ t ^ ">"
 | Edge(t) -> "edge<" ^ string_of_typ t ^ ">"
 | List(t) -> "list<" ^ string_of_typ t ^ ">"

let rec string_of_expr = function
   Ilit(x) -> string_of_int x
 | Slit(x) -> "\"" ^ x ^ "\""
 | Flit(x) -> x
 | Id(x) -> x
 | Unop(o, e) -> string_of_unop o ^ string_of_expr e
 | Binop(e1, o, e2) -> string_of_expr e1 ^ " " ^
   string_of_binop o ^ " " ^ string_of_expr e2
 | Assign(x, e) -> x ^ " = " ^ string_of_expr e
 | AssignField(x, s, e) -> string_of_expr x
     ^ "." ^ s ^ " = " ^ string_of_expr e
 | Call(f, el) ->
   f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
 | Print(e) -> "print(" ^ string_of_expr e ^ ")"
 | Access(x, s) -> string_of_expr x ^ "." ^ s
 | Noexpr -> ""
 | UEdge(n1, n2) -> "(" ^ string_of_expr n1 ^ " <-> "
                         ^ string_of_expr n2 ^ ")"
 | UEdgeC(n1, e, n2) -> "(" ^ string_of_expr n1 ^ " <->[" ^
string_of_expr e
                         ^ "] " ^ string_of_expr n2 ^ ")"
 | DEdge(n1, n2) -> "(" ^ string_of_expr n1 ^ " -> " ^ string_of_expr
n2 ^ ")"
 | DEdgeC(n1, e, n2) -> "(" ^ string_of_expr n1 ^ " -> [" ^
string_of_expr e
                         ^ "] " ^ string_of_expr n2 ^ ")"
 | Index(l, i) -> string_of_expr l ^ "[" ^ string_of_expr i ^ "]"
 | PushBack(l, e) -> string_of_expr l ^ ".push_back("
       ^ string_of_expr e ^ ")"
 | PushFront(l, e) -> string_of_expr l ^ ".push_front("
       ^ string_of_expr e ^ ")"
 | PopBack(l) -> string_of_expr l ^ ".pop_back()"
 | PopFront(l) -> string_of_expr l ^ ".pop_front()"
 | PeekBack(l) -> string_of_expr l ^ ".peek_back()"
 | PeekFront(l) -> string_of_expr l ^ ".peek_front()"
 | AddNode(g, e) -> string_of_expr g ^ ".add_node(" ^ string_of_expr
e ^")"
 | GAdd(g, id, v) -> string_of_expr g ^ ".add("
   ^ string_of_expr id ^ ", " ^  string_of_expr v ^ ")"
 | Contains(g, n) -> string_of_expr g ^ ".contains(" ^ string_of_expr
n ^ ")"
 | ContainsId(g, i) ->
   string_of_expr g ^ ".contains_id(" ^ string_of_expr i ^ ")"
 | AddAll(e, el) -> string_of_expr e ^
     ".add_all(" ^ String.concat ", " (List.map string_of_expr el) ^
")"
```

```
let rec string_of_stmt = function
   Expr(e) -> string_of_expr e ^ ";\n"
 | Block(stmts) ->
   "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\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
(*  | Foreach(t, s, expr, stmt) ->
*)
 | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^
string_of_stmt s
 | Declare(t, lx, Noexpr) -> string_of_typ t ^ " " ^
   (List.fold_left (fun l s -> s ^ "; " ^ l) "" lx ) ^ "\n"
 | Declare(t, ([x] as l), e) when (List.length l) = 1 ->
string_of_typ t ^ " " ^ x ^ "; "
       ^ string_of_expr e ^ ";\n"
 | Declare(_) -> "PARSING ERROR\n"


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

let string_of_fdecl fdecl =
 string_of_typ fdecl.typ ^ " " ^
 fdecl.fname ^ "(" ^
 String.concat ", " (List.map snd fdecl.formals) ^ ")\n{\n" ^
 String.concat ", " (List.map string_of_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)
```

8.4 sast.ml

```
(*
    Semantically Checked Abstract Syntax Tree
    Author: Matthew Sanchez
*)

open Ast

type sexpr = typ * sx
and sx =
    SIlit of int
  | SSlit of string
  | SFlit of string
  | SId of string
  | SUnop of unop * sexpr
  | SBinop of sexpr * binop * sexpr
  | SAssign of string * sexpr
  | SAssignField of sexpr * string * sexpr
  | SCall of string * sexpr list
  | SPrint of sexpr
  | SAccess of sexpr * string
  | SNoexpr
  | SUEdge of sexpr * sexpr
  | SUEdgeC of sexpr * sexpr * sexpr
  | SDEdge of sexpr * sexpr
  | SDEdgeC of sexpr * sexpr * sexpr
  | SIndex of sexpr * sexpr
  | SPushBack of sexpr * sexpr
  | SPushFront of sexpr * sexpr
  | SPopBack of sexpr
  | SPopFront of sexpr
  | SPeekBack of sexpr
  | SPeekFront of sexpr
  | SAddNode of sexpr * sexpr
  | SGAdd of sexpr * sexpr * sexpr
  | SContains of sexpr * sexpr
  | SContainsId of sexpr * sexpr
  | SAddAll of sexpr * sexpr list

type sstmt =
    SExpr of sexpr
```

```
 | SBlock of sstmt list
 | SReturn of sexpr
 | SIf of sexpr * sstmt * sstmt
 | SFor of sexpr * sexpr * sexpr * sstmt
 | SWhile of sexpr * sstmt
 | SDeclare of typ * string list * sexpr
 type sfunc_decl = {
   styp : typ;
   sfname : string;
   sformals : bind list;
   sbody : sstmt list;
 }

type sprogram = bind list * sfunc_decl list

(* Pretty-printing *)

let rec string_of_sexpr (t, e) =
 "(" ^ string_of_typ t ^ " : " ^ (match e with
   SIlit(l) -> string_of_int l
 | SSlit(l) -> "\"" ^ l ^ "\""
 | SFlit(l) -> l
 | SId(s) -> s
 | SUnop(o, e) -> string_of_unop o ^ string_of_sexpr e
 | SBinop(e1, o, e2) ->
     string_of_sexpr e1 ^ " " ^ string_of_binop o ^ " " ^
string_of_sexpr e2
 | SAssign(s, e) -> s ^ " = " ^ string_of_sexpr e
 | SAssignField(x, s, e) -> string_of_sexpr x ^ "."
       ^ s ^ " = " ^ string_of_sexpr e
 | SCall(f, el) ->
     f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^ ")"
 | SPrint(e) -> "print(" ^ string_of_sexpr e ^ ")"
 | SAccess(x, s) -> string_of_sexpr x ^ "." ^ s
 | SNoexpr -> ""
 | SUEdge(n1, n2) -> string_of_sexpr n1 ^ " <-> " ^ string_of_sexpr
n2
 | SUEdgeC(n1, e, n2) -> string_of_sexpr n1 ^ " <-> [" ^
string_of_sexpr e
                         ^ "] " ^ string_of_sexpr n2
 | SDEdge(n1, n2) -> string_of_sexpr n1 ^ " -> " ^ string_of_sexpr n2
 | SDEdgeC(n1, e, n2) -> string_of_sexpr n1 ^ " ->[" ^
string_of_sexpr e
```

```
                          ^ "] " ^ string_of_sexpr n2
 | SIndex(l, i) -> string_of_sexpr l ^ "[" ^ string_of_sexpr i ^ "]"
 | SPushBack(l, e) -> string_of_sexpr l ^ ".push_back("
                      ^ string_of_sexpr e ^ ")"
 | SPushFront(l, e) -> string_of_sexpr l ^ ".push_front("
                      ^ string_of_sexpr e ^ ")"
 | SPopBack(l) -> string_of_sexpr l ^ ".pop_back()"
 | SPopFront(l) -> string_of_sexpr l ^ ".pop_front()"
 | SPeekBack(l) -> string_of_sexpr l ^ ".peek_back()"
 | SPeekFront(l) -> string_of_sexpr l ^ ".peek_front()"
 | SAddNode(g, e) -> string_of_sexpr g ^ ".add_node(" ^
string_of_sexpr e ^")"
 | SGAdd(g, id, v) -> string_of_sexpr g ^ ".add("
   ^ string_of_sexpr id ^ ", " ^ string_of_sexpr v ^ ")"
 | SContains(g, n) ->
   string_of_sexpr g ^ ".contains(" ^ string_of_sexpr n ^ ")"
 | SContainsId(g, i) ->
   string_of_sexpr g ^ ".contains_id(" ^ string_of_sexpr i ^ ")"
 | SAddAll(e, el) -> string_of_sexpr e ^
     ".add_all(" ^ String.concat ", " (List.map string_of_sexpr el) ^
")"

 ) ^ ")"

let rec string_of_sstmt = function
   SExpr(se) -> string_of_sexpr se ^ ";\n"
 | SBlock(ssl) ->
   "{\n" ^ String.concat "" (List.map string_of_sstmt ssl) ^ "}\n"
 | SReturn(se) -> "return " ^ string_of_sexpr se ^ ";\n"
 | SIf(se, ss, SBlock([])) ->
     "if (" ^ string_of_sexpr se ^ ")\n" ^ string_of_sstmt ss
 | SIf(se, ss1, ss2) -> "if (" ^ string_of_sexpr se ^ ")\n" ^
     string_of_sstmt ss1 ^ "else\n" ^ string_of_sstmt ss2
 | SFor(se1, se2, se3, ss) ->
     "for (" ^ string_of_sexpr se1  ^ " ; " ^ string_of_sexpr se2 ^ "
; " ^
     string_of_sexpr se3  ^ ") " ^ string_of_sstmt ss
 | SWhile(se, ss) -> "while (" ^ string_of_sexpr se ^ ") " ^
     string_of_sstmt ss
 | SDeclare(t, lx, (_,SNoexpr)) -> string_of_typ t ^ " " ^
   (List.fold_left (fun l s  -> s ^ "; " ^ l) "" lx ) ^ "\n"
 | SDeclare(t, ([x] as l), e) when (List.length l) = 1 ->
string_of_typ t ^ " " ^ x ^ "; "
```

```
        ^ string_of_sexpr e ^ ";\n"
 | SDeclare(_) -> "PARSING ERROR\n"

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

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

8.5 semant.ml

```
(*
    Semantic checking, produces sast from ast
    Author: Ashar Nadeem and Matthew Sanchez
*)

open Ast
open Sast

module StringMap = Map.Make(String)

let valid_element_type = function
            (Void,_) -> raise(Failure("error: list of type void not
allowed"))
      | _ -> ()

let check_type (ex, ty) =
      if ex = ty then ()
      else raise (Failure ("error: "))

let get_type(t, _) = t

let first_element (myList) = match myList with
 [] -> Void
| first_e1 :: _ -> get_type(first_e1)

(* Check global variables, then each function *)
```

```
let check (globals, functions) =

  (* Check global variables *)
  (* No void bindings or duplicate names *)
  let check_binds (kind : string) (binds: bind list) =
    List.iter (function
        (Void, b) -> raise (Failure ("error: illegal void " ^ kind ^
        " " ^ b))
      | _ -> ()) binds;

    let rec dups = function
        [] -> ()
      | ((_,n1) :: (_,n2) :: _) when n1 = n2 -> raise(Failure
          ("error: variable previously declared: " ^ n1))
      | _ :: t -> dups t
    in dups (List.sort (fun (_,a) (_,b) -> compare a b) binds)
  in check_binds "global" globals;

  (* built-in decls go here (none at the moment) *)
  let built_in_decls =
    let add_bind map (ft, name, t) = StringMap.add name {
      typ = ft;
      fname = name;
      formals = [t];
      body = [] } map
      in List.fold_left add_bind StringMap.empty [    ]


  in
  (* adds unique function names to symbol table *)
  let add_func map fd =
    let built_in_err = "function " ^ fd.fname ^ " may not be defined"
    and dup_err = "error: function previously declared" ^ fd.fname
    and make_err er = raise (Failure er)
    and n = fd.fname
    in match fd with
        _ 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
  (* Collects built-in and declared functions *)
  let function_decls = List.fold_left add_func built_in_decls
```

```
functions
  in

  (* Finds a function in the table *)
  let find_func s =
    try StringMap.find s function_decls
    with Not_found -> raise (Failure ("error: function never
declared: " ^ s))
  in

  (* Require main function for all programs *)
  let _ = find_func "main" in

  let check_function func =
    (* No duplicates in formals *)
    check_binds "formal" func.formals;

    let check_assign lvaluet rvaluet err =
      if lvaluet = rvaluet then lvaluet else raise (Failure (err ^
""))
    in

    (* Since we aren't using locals, find declarations in fcn *)
    let rec concat_statements locals = function
        [] -> locals
      | h::t -> concat_statements (match h with
                  Declare(t, x, Noexpr) ->
                    List.fold_left (fun l s -> (t, s)::l) locals x
                | Declare(t, ([x] as l), _) when List.length l = 1
                    -> (t, x) :: locals
                | _ -> locals) t
    in

    let fulldecls = globals @ func.formals @ concat_statements []
func.body

    in  (List.iter (fun (t, _) -> match t with
                  Node(it) -> (match it with
                      List(_) | Node(_) | Graph(_) | Edge(_) | Void
->
                        raise (Failure ("error: nodes cannot wrap "
                               ^ string_of_typ it))
                    | _ -> ())
```

```
                 | Edge(it) -> (match it with
                     List(_) | Node(_) | Graph(_) | Edge(_) | Void
->
                       raise (Failure ("error: edges cannot wrap "
                             ^ string_of_typ it))
                     | _ -> ())
                 | Graph(it) -> (match it with
                     List(_) | Node(_) | Edge(_) | Graph(_) | Void
->
                       raise (Failure ("error: graphs cannot wrap "
                             ^ string_of_typ it))
                     | _ -> ())
                 | List(Void) -> raise (Failure
                   "error: lists cannot wrap void")
                 | _ -> ())
                    fulldecls);
    (* Local symbol table for function *)

    (* check for duplicate ids *)
    let check_decls (kind : string) (binds: bind list) =
    List.iter (function
        (Void, b) -> raise (Failure ("error: illegal void " ^ kind ^
        " " ^ b))
      | _ -> ()) binds;

    let rec dups = function
        [] -> ()
      | ((_,n1) :: (_,n2) :: _) when n1 = n2 -> raise(Failure
          ("error: variable previously declared: " ^ n1))
      | _ :: t -> dups t
    in dups (List.sort (fun (_,a) (_,b) -> compare a b) fulldecls)
  in check_decls "global" globals;

    let symbols = List.fold_left (fun m (t, x) -> StringMap.add x t
m)
      StringMap.empty (fulldecls)
    in

    (* Gets symbols from table *)
    let type_of_identifier s =
      try StringMap.find s symbols
      with Not_found -> raise (Failure ("error: variable never
declared: " ^ s))
```

```
    in

    (* Checks expressions *)
    let rec expr =
      function
        Ilit l -> (Int, SIlit l)
      | Slit l -> (String, SSlit l)
      | Flit l -> (Float, SFlit l)
      | Id s -> (type_of_identifier s, SId s)
      | Unop(o, e) as ex ->
          let (t, e') = expr e in
          let ty = match o with
            Neg when t = Int || t = Float -> t
          | Not when t = Int -> Int
          | _ -> raise (Failure ("error: illegal unary operator " ^
                                 string_of_unop o ^ string_of_typ t ^
                                 " in " ^ string_of_expr ex))
          in (ty, SUnop(o, (t, e')))
      | Binop(e1, o, e2) as ex ->
          let (t1, e1') = expr e1 and (t2, e2') = expr e2 in
          let same = t1 = t2 in
          let ty = match o with
            Add | Sub | Mul | Div | Mod when same && t1 = Int -> Int
          | Add | Sub | Mul | Div | Mod when same && t1 = Float ->
Float
          | Eq | Neq  when same -> (match t1 with
              Int | Float | String |
              Node(_) | Edge(_) | List(_) | Graph (_) -> Int
            | _ -> raise (Failure "error: illegal comparison"))
          | Less | Leq | Greater | Geq
                   when same && (t1 = Int || t1 = Float) -> Int
          | And | Or when same && t1 = Int -> Int
          | _ -> raise (Failure("error: illegal binary operator" ^
                        string_of_typ t1 ^ " " ^ string_of_binop o ^
" " ^
                        string_of_typ t2 ^ " in " ^ string_of_expr
ex))
          in (ty, SBinop((t1,e1'), o, (t2, e2')))
      | Assign(x, e) as ex ->
          let lt = type_of_identifier x
          and (rt, e') = expr e in
          let err = "error: illegal HERE assignment " ^ string_of_typ
lt ^ " = " ^
```

```
                    string_of_typ rt ^ " in " ^ string_of_expr ex
          in (check_assign lt rt err, SAssign(x, (rt, e')))
      | AssignField(x, s, e) as ex -> let sx = expr x in
            let (lt, it) = (match sx with
              (Node(t), _) -> t, List(Int)
            | _ -> raise (Failure ("error: cannot access this
type")))
          and (rt, e') = expr e in
          let err = "error1111: illegal assignment " ^ string_of_typ
lt ^ " = "
                  ^ string_of_typ rt ^ " in " ^ string_of_expr ex
          in (match s with
              "val" ->
                (check_assign lt rt err, SAssignField(sx, s, (rt,
e')))
            | "id" ->
                (check_assign Int rt err, SAssignField(sx, s, (rt,
e')))
            | "edges" ->
                (check_assign it rt err, SAssignField(sx, s, (rt,
e')))
            | _ -> raise (Failure (err)))
      | Call(f, el) as call ->
          let fd = find_func f in
          let param_length = List.length fd.formals in
          if List.length el != param_length then
            raise (Failure ("error: 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 = "error: 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 el
          in (fd.typ, SCall(f, args))
      | Print(e) -> let e' = expr e in (match e' with
            (Void, _) -> raise (Failure "print used on void")
          | _ -> (Void, SPrint(e')))
      | Access(x, s) -> let sx = expr x in (match sx with
            (Node(tn), _) -> (match s with
                "val" -> (tn, SAccess(sx, s))
              | "id"  -> (Int, SAccess(sx, s))
              | "edges" -> (List(Edge(tn)), SAccess(sx, s))
              | _ -> raise (Failure ("error: invalid node field " ^
s)))
          | (List(_), _) -> (match s with
                "size" -> (Int, SAccess(sx, s))
              | _ -> raise (Failure ("error: invalid list field " ^
s)))
          | (Graph(tg), _) -> (match s with
                "size" -> (Int, SAccess(sx, s))
              | "nodes" -> (List(Node(tg)), SAccess(sx, s))
              | "root" -> (Node(tg), SAccess(sx, s))
              | _ -> raise (Failure ("error: invalid graph field " ^
s)))
          | (Edge(te), _) -> (match s with
                "weight" -> (te, SAccess(sx, s))
              | "dest" -> (Node(te), SAccess(sx, s))
              | "t" -> (Int, SAccess(sx, s))
              | _ -> raise (Failure ("error: invalid edge field " ^
s)))
          | _ -> raise
              (Failure ("error: this type does not have this field: "
^ s)))
      | Noexpr -> (Void, SNoexpr)
      | UEdge(n1, n2) -> let sn1 = expr n1 and
                             sn2 = expr n2 in (match (sn1, sn2) with
            ((Node(a), _),  (Node(b), _)) when a = b ->
                    (Node(a), SUEdge(sn1, sn2))
          | _ -> raise (Failure ("error: UEdge fail")))

      | UEdgeC(n1, e, n2) -> let sn1 = expr n1 and
                                sn2 = expr n2 in (match (sn1, sn2)
with
            ((Node(a), _),  (Node(b), _)) when a = b ->
                    (Node(a), SUEdgeC(sn1, expr e, sn2))
          | _ -> raise (Failure ("error: UEdgeC fail")))

      | DEdge(n1, n2) -> let sn1 = expr n1 and
                             sn2 = expr n2 in (match (sn1, sn2) with
            ((Node(a), _),  (Node(b), _)) when a = b ->
                    (Node(a), SDEdge(sn1, sn2))
          | _ -> raise (Failure ("error: DEdge fail")))

      | DEdgeC(n1, e, n2) -> let sn1 = expr n1 and
                                 sn2 = expr n2 in (match (sn1, sn2)
with
            ((Node(a), _),  (Node(b), _)) when a = b ->
              (Node(a), SDEdgeC(sn1, expr e, sn2))
          | _ -> raise (Failure ("error: DEdgeC fail")))
      | Index(l, i) -> let l' = expr l and
                              i' = expr i in (match (l', i') with
            ((List(t),_), (Int, _)) -> (t, SIndex(l', i'))
          | ((Graph(t),_), (Int, _)) -> (Node(t), SIndex(l', i'))
          | (_, (Int,_)) -> raise (Failure
          ("error: cannot index non-list/graph " ^ string_of_expr l))
          | ((List(_),_), _) -> raise (Failure
          ("error: cannot index with non-int " ^ string_of_expr i))
          | _ -> raise (Failure ("error: invalid index of "
          ^ string_of_expr l ^ " with " ^ string_of_expr i)))
      | PushBack(l, e) -> let l' = expr l
                         and e' = expr e in (match (l', e') with
          ((List(tl),_), (te, _)) when tl = te ->
                                 (List(tl), SPushBack(l', e'))
        | ((List(tl),_), _) -> raise (Failure ("error: cannot push "
        ^ string_of_expr e ^ " to list of type " ^ string_of_typ tl))
        | _ -> raise (Failure ("error: push_back on non-list "
          ^ string_of_expr e)))
      | PushFront(l, e) -> let l' = expr l
                           and e' = expr e in (match (l', e') with
          ((List(tl),_), (te, _)) when tl = te ->
                                 (List(tl), SPushFront(l', e'))
        | ((List(tl),_), _) -> raise (Failure ("error: cannot push "
        ^ string_of_expr e ^ " to list of type " ^ string_of_typ tl))
        | _ -> raise (Failure ("error: push_back on non-list "
        ^ string_of_expr e)))
      | PopBack(l) -> let l' = expr l in (match l' with
            (List(tl),_) -> (tl, SPopBack(l'))
          | _ -> raise (Failure ("error: pop_back on non-list "
          ^ string_of_expr l)))
      | PopFront(l) -> let l' = expr l in (match l' with
            (List(tl),_) -> (tl, SPopFront(l'))
          | _ -> raise (Failure ("error: pop_front on non-list "
          ^ string_of_expr l)))
      | PeekBack(l) -> let l' = expr l in (match l' with
            (List(tl),_) -> (tl, SPeekBack(l'))
          | _ -> raise (Failure ("error: peek_back on non-list "
          ^ string_of_expr l)))
      | PeekFront(l) -> let l' = expr l in (match l' with
            (List(tl),_) -> (tl, SPeekFront(l'))
          | _ -> raise (Failure ("error: peek_front on non-list "
          ^ string_of_expr l)))
      | AddNode(g, e) -> let g' = expr g  and e' = expr e
        in (match (g', e') with
          ((Graph(tg), _), (Node(tn), _)) ->
            if tg = tn then
            (Void, SAddNode(g', e'))
            else raise (Failure ("error: graph and node types do not
match,"
            ^ string_of_typ tg ^ " vs " ^ string_of_typ tn))
        | ((Graph(_), _), _) ->
            raise (Failure "error: cannot add_node non-node to
graph")
        | (_, _) ->
            raise (Failure "error: cannot add_node to non-graph"))
      | GAdd(g, id, v) -> let g' = expr g and id' = expr id and v' =
expr v
        in (match (g', id', v') with
          ((Graph(tg), _), (Int, _), (tn, _)) ->
          if tg = tn then (Void, SGAdd(g', id', v'))
          else raise (Failure ("error: adding mismatched type "
          ^ string_of_typ tn ^ " to graph of " ^ string_of_typ tg))
        | ((Graph(_), _), _,_) ->
          raise (Failure "error: invalid parameters to add")
        | (_, _, _) ->
          raise (Failure "error: cannot add to non-graph"))
      | Contains(g, n) -> let g' = expr g and n' = expr n
      in (match (g', n' ) with
          ((Graph(tg),_), (Node(tn), _)) ->
            if tg = tn then (Int, SContains(g', n'))
            else raise (Failure "error: type mismatch on contains")
        | ((Graph(_),_), _) ->
            raise (Failure "error: contains arg must be node")
        | (_, _) -> raise (Failure "error: contains used on
non-graph"))
      | ContainsId(g, id) -> let g' = expr g and id' = expr id
      in (match (g', id' ) with
          ((Graph(_),_), (Int, _)) -> (Int, SContainsId(g', id'))
        | ((Graph(_),_), _) ->
            raise (Failure "error: contains_id arg must be int")
        | (_, _) -> raise (Failure "error: contains_id used on
non-graph"))
      | AddAll(e, el) -> let e' = expr e in (match e' with
          (List(t),_) -> let check_arg arg = let arg' = expr arg in
                (match arg' with
              (ta, _) when ta = t-> arg'
            | _ -> raise (Failure "error: mismatch arg type in
add_all"))
            in let args = List.map check_arg el in (List(t),
SAddAll(e', args))
        | (Graph(t), _) -> let check_arg arg = let arg' = expr arg in
                (match arg' with
              (Node(ta), _) when ta = t-> arg'
            | _ -> raise (Failure "error: mismatch arg type in
add_all"))
            in let args = List.map check_arg el in (List(t),
SAddAll(e', args))
        | _ -> raise (Failure "error: addall not used on list or
graph"))

    in

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

    (* Check statements *)
    let rec check_stmt = function
        Expr e -> SExpr (expr e)
      | 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)
            | s :: ss -> check_stmt s :: check_stmt_list ss
            | [] -> []
          in SBlock(check_stmt_list sl)
      | Return e -> let (t, e') = expr e in
          if t = func.typ then SReturn (t, e')
          else raise (Failure
              ("error: expected return to be of type: " ^
string_of_typ t))
      | If(p, b1, b2) -> SIf(check_bool_expr p, check_stmt b1,
check_stmt b2)
      | For(e1, e2, e3, s) ->
          SFor(expr e1, expr e2, expr e3, check_stmt s)
      | While(p, s) -> SWhile(check_bool_expr p, check_stmt s)
      | Declare(t, x, Noexpr) -> SDeclare(t, x, (Void, SNoexpr))
      | Declare(t, l, e) as d when List.length l = 1
          -> let (t', v) = expr e in
          ignore v;
          if t' = t then SDeclare(t, l, (t', v))
          else raise (Failure ("error: assignment does not match
value in " ^
                string_of_stmt d ))
      | Declare(_) -> raise (Failure ("error: invalid declaration"))
    in
    (* check_function finally complete *)
    { styp = func.typ;
      sfname = func.fname;
      sformals = func.formals;
      sbody = match check_stmt (Block func.body) with
        SBlock(sl) -> sl
        | _ -> raise (Failure ("internal error"))
    }
  in (globals, List.map check_function functions)
```

8.6 codegen.ml

```
(*
    Generates LLVM IR from graphene input code (in the form of sast)
    Author: Ashar Nadeem and Matthew Sanchez
*)

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

```
module StringMap = Map.Make(String)

let get_type(t, _) = t

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

 let llmem_graph = L.MemoryBuffer.of_file "graphene.bc" in
 let llm_graph = Llvm_bitreader.parse_bitcode context llmem_graph in
  (* Create the LLVM compilation module into which
    we will generate code *)
 let the_module = L.create_module context "Graphene" in

 (* Get types from the context *)
 let i32_t      = L.i32_type    context
 and i8_t       = L.i8_type     context
 and string_t   = L.pointer_type (L.i8_type context)
 and float_t    = L.double_type context
 and void_t     = L.void_type   context
 and void_ptr_t = L.pointer_type (L.i8_type context)
 and lst_t      = L.pointer_type
   (match L.type_by_name llm_graph "struct.list" with
     None -> raise (Failure "error: missing implementation for struct
list")
   | Some t -> t)
 and node_t  = L.pointer_type
   (match L.type_by_name llm_graph "struct.node" with
     None -> raise (Failure "error: missing implementation for struct
node")
   | Some t -> t)
 and edge_t  = L.pointer_type
   (match L.type_by_name llm_graph "struct.edge" with
     None -> raise (Failure "error: missing implementation for struct
edge")
   | Some t -> t)
 and graph_t      = L.pointer_type
   (match L.type_by_name llm_graph "struct.graph" with
     None -> raise (Failure "error: missing implementation for struct
graph")
   | Some t -> t)
 in
  (* Return the LLVM type for a Graphene type *)
 let ltype_of_typ = function
     A.Int      -> i32_t
   | A.String   -> string_t
   | A.Float    -> float_t
   | A.Void     -> void_t
   | A.Node _   -> node_t
   | A.List _   -> lst_t
   | A.Edge _   -> edge_t
   | A.Graph _  -> 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
       | A.Int -> L.const_int (ltype_of_typ t) 0
       | _ -> raise (Failure
         "error: this type cannot be declared globally")
     in StringMap.add n (L.define_global n init the_module) m in
   List.fold_left global_var StringMap.empty globals in

 let printf_t : L.lltype =
     L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
 let printf_f : L.llvalue =
     L.declare_function "printf" printf_t the_module in

 let list_init_t : L.lltype =
     L.function_type lst_t [| |] in
 let list_init_f : L.llvalue =
     L.declare_function "list_init" list_init_t the_module in

 let node_init_t : L.lltype =
     L.function_type node_t [| |] in
 let node_init_f : L.llvalue =
     L.declare_function "node_init" node_init_t the_module in

 let edge_init_t : L.lltype =
     L.function_type edge_t [| void_ptr_t ; node_t; i32_t |] in
 let edge_init_f : L.llvalue =
     L.declare_function "edge_init" edge_init_t the_module in
 let graph_init_t : L.lltype =
     L.function_type graph_t [| |] in
 let graph_init_f : L.llvalue =
     L.declare_function "graph_init" graph_init_t the_module in

 let list_index_t : L.lltype =
     L.function_type void_ptr_t [|lst_t ; i32_t|] in
 let list_index_f : L.llvalue =
     L.declare_function "list_index" list_index_t the_module in

 let list_push_back_t : L.lltype =
     L.function_type i32_t [| lst_t ; void_ptr_t |] in
 let list_push_back_f : L.llvalue =
     L.declare_function "list_push_back" list_push_back_t the_module
in

 let list_push_front_t : L.lltype =
     L.function_type i32_t [| lst_t ; void_ptr_t |] in
 let list_push_front_f : L.llvalue =
     L.declare_function "list_push_front" list_push_front_t
the_module in

 let list_pop_back_t : L.lltype =
     L.function_type void_ptr_t [| lst_t |] in
 let list_pop_back_f : L.llvalue =
     L.declare_function "list_pop_back" list_pop_back_t the_module in

 let list_pop_front_t : L.lltype =
     L.function_type void_ptr_t [| lst_t |] in
 let list_pop_front_f : L.llvalue =
     L.declare_function "list_pop_front" list_pop_front_t the_module
in

 let list_peek_back_t : L.lltype =
     L.function_type void_ptr_t [| lst_t |] in
 let list_peek_back_f : L.llvalue =
     L.declare_function "list_peek_back" list_peek_back_t the_module
in

 let list_peek_front_t : L.lltype =
     L.function_type void_ptr_t [| lst_t |] in
 let list_peek_front_f : L.llvalue =
     L.declare_function "list_peek_front" list_peek_front_t
the_module in

 let graph_add_node_t : L.lltype =
     L.function_type i32_t [| graph_t ; node_t |] in
 let graph_add_node_f : L.llvalue =
     L.declare_function "graph_add_node" graph_add_node_t the_module
in

 let graph_add_t : L.lltype =
     L.function_type node_t [| graph_t; i32_t; void_ptr_t |] in
 let graph_add_f : L.llvalue =
     L.declare_function "graph_add" graph_add_t the_module in

 let graph_get_node_t : L.lltype =
     L.function_type node_t [| graph_t ; i32_t |] in
 let graph_get_node_f : L.llvalue =
     L.declare_function "graph_get_node" graph_get_node_t the_module
in
  let graph_contains_node_t : L.lltype =
     L.function_type i32_t [| graph_t; node_t |] in
 let graph_contains_node_f : L.llvalue =
     L.declare_function "graph_contains_node" graph_contains_node_t
the_module
 in
 let graph_contains_id_t : L.lltype =
     L.function_type i32_t [| graph_t; i32_t |] in
 let graph_contains_id_f : L.llvalue =
     L.declare_function "graph_contains_id" graph_contains_id_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 str_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
       let rec gather_locals locals = function
           [] -> locals
         | h::t -> gather_locals (match h with
            SDeclare(t, s, (_, Sast.SNoexpr)) ->
     List.fold_left (fun locs sl -> (t, sl) :: locs) locals s
           | SDeclare(t, ([s] as l), _) when List.length l = 1
```

```
                          -> (t, s) :: locals
               | _ -> locals) t
        in
        let complete_locals = gather_locals [] fdecl.sbody
        in
        List.fold_left add_local formals complete_locals
      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 ((_, e) : sexpr) = match e with
        SIlit i     -> L.const_int i32_t i
      | SFlit l     -> L.const_float_of_string float_t l
      | SSlit s     -> L.build_global_stringptr (s ^ "\x00") "str"
builder
      | SNoexpr     -> L.const_int i32_t 0
      | SId s       -> L.build_load (lookup s) s builder
      | SAssign (s, e) -> let e' = expr builder e in
                          ignore(L.build_store e' (lookup s) builder);
e'
      | SAssignField (x, s, e) -> let e' = expr builder e and
                                      x' = expr builder x in
        (match x with
          (A.Node(t), _)  ->
          (match s with
            "id" ->
            let p = L.build_struct_gep x' 0 "struct.ptr" builder in
          ignore(L.build_store e' p builder); e'
          | "val" ->
          let str_ptr = L.build_struct_gep x' 1 "struct.ptr" builder
in
          let void_ptr = L.build_load str_ptr "void.ptr" builder in
          let cast =
            L.build_bitcast void_ptr (L.pointer_type (ltype_of_typ t))
            "val" builder in
          ignore (L.build_store e' cast builder); e'
          | "edges" ->
            let str_ptr = L.build_struct_gep x' 2 "struct.ptr" builder
in
```

```
           let edge_ptr = L.build_load str_ptr "edge.ptr" builder in
           ignore(L.build_store e' edge_ptr builder); e'
          | _ -> raise (Failure
               "this should never happen, field error missed in
semant"))
        | _ -> raise (Failure "internal error on assignfield"))
      | 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.Mul     -> L.build_fmul
          | A.Div     -> L.build_fdiv
          | A.Mod     -> L.build_frem
          | A.Eq      -> L.build_fcmp L.Fcmp.Oeq
          | A.Neq     -> L.build_fcmp L.Fcmp.One
          | A.Less    -> L.build_fcmp L.Fcmp.Olt
          | A.Leq     -> L.build_fcmp L.Fcmp.Ole
          | A.Greater -> L.build_fcmp L.Fcmp.Ogt
          | A.Geq     -> L.build_fcmp L.Fcmp.Oge
          | A.And | A.Or ->
             raise (Failure
               "internal error: semant should have rejected and/or on
float")
          ) e1' e2' "tmp" builder
      | SBinop ((A.Int,_ ) as e1, op, e2) ->
          let e1' = expr builder e1
          and e2' = expr builder e2 in
          (match op with
           A.Add     -> L.build_add
          | A.Sub     -> L.build_sub
          | A.Mul     -> L.build_mul
          | A.Div     -> L.build_sdiv
          | A.Mod     -> L.build_srem
          | A.And     -> L.build_and
          | A.Or      -> L.build_or
          | A.Eq      -> L.build_icmp L.Icmp.Eq
          | A.Neq     -> L.build_icmp L.Icmp.Ne
          | A.Less    -> L.build_icmp L.Icmp.Slt
          | A.Leq     -> L.build_icmp L.Icmp.Sle
          | A.Greater -> L.build_icmp L.Icmp.Sgt
          | A.Geq     -> L.build_icmp L.Icmp.Sge
          ) e1' e2' "tmp" builder
            | 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
      | SBinop ((ty,_ ) as e1, op, e2) ->
          if op = A.Eq then
            let e1' = expr builder e1
            and e2' = expr builder e2 in
            L.build_icmp L.Icmp.Eq e1' e2' "tmp" builder
          else if op = A.Neq then
            let e1' = expr builder e1
            and e2' = expr builder e2 in
            L.build_icmp L.Icmp.Ne e1' e2' "tmp" builder
          else raise (Failure ("invalid operator " ^ A.string_of_binop
op ^
            " on " ^ A.string_of_typ ty))
      | SIndex ((lt, _) as s, e) -> (match lt with
          A.List(t) -> (match t with
            A.Int | A.Float ->
          let ptr = (L.build_call list_index_f
            [|expr builder s; expr builder e |] "list_index" builder)
            and ty = (L.pointer_type (ltype_of_typ t)) in
          let cast = L.build_bitcast ptr ty "cast" builder in
          L.build_load cast "val" builder
          | _ ->  let ptr = L.build_call list_index_f
            [|expr builder s; expr builder e |] "list_index" builder
            and ty =  (ltype_of_typ t) in
             L.build_bitcast ptr ty "cast" builder)
        | A.Graph(_)->
          L.build_call graph_get_node_f
          [| expr builder s; expr builder e |] "graph_get_node"
builder
        | _ -> raise (Failure
          ("this should never happen, error in semant")) )
      | 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
      | SPrint(e) -> (match e with
          (A.Int, _) ->
            L.build_call printf_f [| int_format_str ; (expr builder e)
|]
            "printf" builder
        | (A.Float, _) ->
            L.build_call printf_f [| float_format_str ; (expr builder
e) |]
            "printf" builder
        | (A.String, _) ->
            L.build_call printf_f [| str_format_str ; (expr builder e)
|]
            "printf" builder
        | (A.Node(t), _) ->
          let id = expr builder ((A.Int), (SAccess(e, "id")))
          in let nval = (expr builder (t, SAccess(e, "val")))
          in let edges = (expr builder
            (A.Int, SAccess((A.List(A.Edge(t)), SAccess(e, "edges")),
"size")))
           in  let str_mod  =
          (match t with
            A.Int -> "d"
          | A.Float -> "f"
          | A.String -> "s"
          | _ -> raise (Failure "internal error on node") )
          in let format_str =
            L.build_global_stringptr
            ("id: %d, val: %" ^ str_mod ^ ", edges: %d\n") "fmt"
builder in
            L.build_call printf_f
            [| format_str; id; nval; edges|]
            "printf" builder
        | _ -> raise (Failure "internal error on print") )
      | SAccess (s, x) -> let s' = expr builder s in
          (match s with
          (A.Node(t), _) ->
            (match x with
              "id" ->
              (let p = L.build_struct_gep s' 0 "struct.ptr" builder in
            L.build_load p ("struct.val." ^ x) builder)
            | "val" ->
            (let str_ptr = L.build_struct_gep s' 1 "struct.ptr"
 builder in
            let void_ptr = L.build_load str_ptr "void.ptr" builder in
            let ty = ltype_of_typ t in
            let cast =
              L.build_bitcast void_ptr (L.pointer_type ty) "cast"
 builder in
            L.build_load cast ("struct.val." ^ x ^ ".value") builder)
            | "edges" ->
              (let str_ptr = L.build_struct_gep s' 2 "struct.ptr"
 builder in
              L.build_load str_ptr ("struct.val." ^ x ) builder)
            | _ -> raise (Failure
                ("this should never happen, error in semant")))
        | (A.List(_), _) ->
            (match x with
              "size" -> (let p = L.build_struct_gep s' 0 "struct.ptr"
 builder
              in L.build_load p ("struct.val." ^ x) builder)
            | _ -> raise
                (Failure "this should never happen, error in semant")
              )
        | (A.Graph(_), _) -> let mem = (match x with
            | "nodes" -> 0
            | "root" -> 1
            | _ -> raise (Failure "internal error in accessing")) in
          let p = L.build_struct_gep s' mem "struct.ptr" builder
          in L.build_load p ("struct.val." ^ x) builder
        | (A.Edge(t), _) -> (match x with
              "weight" ->
               (let str_ptr = L.build_struct_gep s' 0 "struct.ptr"
 builder in
            let void_ptr = L.build_load str_ptr "void.ptr" builder in
            let ty = ltype_of_typ t in
            let cast =
              L.build_bitcast void_ptr (L.pointer_type ty) "cast"
 builder in
            L.build_load cast ("struct.val." ^ x ^ ".value") builder)
            | "dest" ->
                let str_ptr = L.build_struct_gep s' 1 "struct.ptr"
 builder
                in
                  L.build_load str_ptr ("struct.val." ^ x) builder
            | "t" ->
                let str_ptr = L.build_struct_gep s' 2 "struct.ptr"
 builder
                in L.build_load str_ptr ("struct.val." ^ x) builder
            | _ -> raise (Failure "internal error in accessing"))
        | _ -> raise (Failure "internal error")
          )
      | SDEdge (n1, n2) -> let n1p = expr builder n1 and
                               n2p = expr builder n2 in
        let atr =
        (match n1 with
          (A.Node(A.String), _) ->
            ((L.build_global_stringptr ("\x00") "str" builder),
 A.String)
        | (A.Node(A.Int), _) ->
            ((L.const_int i32_t 0), A.Int)
        | (A.Node(A.Float), _ )->
            ((L.const_float float_t 0.0), A.Float)
        | _ -> raise (Failure "internal error"))
        in
        (* allocate space for and store weight value *)
        let ptr = L.build_malloc (ltype_of_typ (snd atr)) "element"
 builder in
        ignore (L.build_store (fst atr) ptr builder);
        (* cast to void * *)
        let cast = L.build_bitcast ptr void_ptr_t "cast" builder in
        (* call edge init function, returns pointer to edge *)
         let fedgep = L.build_call edge_init_f
          [|cast; n2p; L.const_int i32_t 1 |]
          "edge_init" builder and
            bedgep = L.build_call edge_init_f
          [|cast; n1p; L.const_int i32_t 0|]
          "edge_init" builder in
          (* get edge lists for nodes *)
        let n1el = L.build_struct_gep n1p 2 "struct.ptr" builder and
            n2el = L.build_struct_gep n2p 2 "struct.ptr" builder in
        let n1ell = L.build_load n1el "temp" builder and
            n2ell = L.build_load n2el "temp" builder in
            (* cast edge ptrs to void * *)
        let cast1 = L.build_bitcast fedgep void_ptr_t
            "cast" builder and
            cast2 = L.build_bitcast bedgep void_ptr_t
            "cast" builder in
```

```
            (* push edge ptrs to node edge lists *)
        ignore (L.build_call list_push_back_f
            [| n1ell; cast1|] "edge_push" builder);
        ignore (L.build_call list_push_back_f
            [| n2ell; cast2|] "edge_push" builder);
            expr builder n1;
      | SDEdgeC (n1, e, n2) -> let e' = expr builder e in
          let n1p = expr builder n1 and
              n2p = expr builder n2 in
          let typ =
        (match n1 with
          (A.Node(A.String), _) -> (A.String)
        | (A.Node(A.Int), _) ->  (A.Int)
        | (A.Node(A.Float), _ )->  (A.Float)
        | _ -> raise (Failure "internal error"))
        in
        let ptr = L.build_malloc (ltype_of_typ (typ)) "element"
builder in
        ignore (L.build_store e' ptr builder);
        let cast = L.build_bitcast ptr void_ptr_t "cast" builder in
         let fedgep = L.build_call edge_init_f
          [|cast; n2p; L.const_int i32_t 1 |]
          "edge_init" builder and
            bedgep = L.build_call edge_init_f
          [|cast; n1p; L.const_int i32_t 0|]
          "edge_init" builder in
        let n1el = L.build_struct_gep n1p 2 "struct.ptr" builder and
            n2el = L.build_struct_gep n2p 2 "struct.ptr" builder in
        let n1ell = L.build_load n1el "temp" builder and
            n2ell = L.build_load n2el "temp" builder in
        let cast1 = L.build_bitcast fedgep void_ptr_t
            "cast" builder and
            cast2 = L.build_bitcast bedgep void_ptr_t
            "cast" builder in
        ignore (L.build_call list_push_back_f
            [| n1ell; cast1|] "edge_push" builder);
        ignore (L.build_call list_push_back_f
            [| n2ell; cast2|] "edge_push" builder);
            expr builder n1;
      | SUEdge (n1, n2) -> let n1p = expr builder n1 and
                                n2p = expr builder n2 in
        let atr =
        (match n1 with
          (A.Node(A.String), _) ->
            ((L.build_global_stringptr ("\x00") "str" builder),
 A.String)
        | (A.Node(A.Int), _) ->
            ((L.const_int i32_t 0), A.Int)
        | (A.Node(A.Float), _ )->
            ((L.const_float float_t 0.0), A.Float)
        | _ -> raise (Failure "internal error"))
        in
        let ptr = L.build_malloc (ltype_of_typ (snd atr)) "element"
 builder in
        ignore (L.build_store (fst atr) ptr builder);
        let cast = L.build_bitcast ptr void_ptr_t "cast" builder in
         let fedgep = L.build_call edge_init_f
          [|cast; n2p; L.const_int i32_t 1 |]
          "edge_init" builder and
            bedgep = L.build_call edge_init_f
          [|cast; n1p; L.const_int i32_t 1|]
          "edge_init" builder in
        let n1el = L.build_struct_gep n1p 2 "struct.ptr" builder and
            n2el = L.build_struct_gep n2p 2 "struct.ptr" builder in
        let n1ell = L.build_load n1el "temp" builder and
            n2ell = L.build_load n2el "temp" builder in
        let cast1 = L.build_bitcast fedgep void_ptr_t
            "cast" builder and
            cast2 = L.build_bitcast bedgep void_ptr_t
            "cast" builder in
        ignore (L.build_call list_push_back_f
            [| n1ell; cast1|] "edge_push" builder);
        ignore (L.build_call list_push_back_f
            [| n2ell; cast2|] "edge_push" builder);
            expr builder n1;
      | SUEdgeC (n1, e, n2) -> let e' = expr builder e in
          let n1p = expr builder n1 and
              n2p = expr builder n2 in
          let typ =
        (match n1 with
          (A.Node(A.String), _) -> (A.String)
        | (A.Node(A.Int), _) ->  (A.Int)
        | (A.Node(A.Float), _ )->  (A.Float)
        | _ -> raise (Failure "internal error"))
        in
        let ptr = L.build_malloc (ltype_of_typ (typ)) "element"
```

```
builder in
       ignore (L.build_store e' ptr builder);
       let cast = L.build_bitcast ptr void_ptr_t "cast" builder in
        let fedgep = L.build_call edge_init_f
         [|cast; n2p; L.const_int i32_t 1 |]
         "edge_init" builder and
           bedgep = L.build_call edge_init_f
         [|cast; n1p; L.const_int i32_t 1|]
         "edge_init" builder in
       let n1el = L.build_struct_gep n1p 2 "struct.ptr" builder and
           n2el = L.build_struct_gep n2p 2 "struct.ptr" builder in
       let n1ell = L.build_load n1el "temp" builder and
           n2ell = L.build_load n2el "temp" builder in
       let cast1 = L.build_bitcast fedgep void_ptr_t
           "cast" builder and
           cast2 = L.build_bitcast bedgep void_ptr_t
           "cast" builder in
       ignore (L.build_call list_push_back_f
           [| n1ell; cast1|] "edge_push" builder);
       ignore (L.build_call list_push_back_f
           [| n2ell; cast2|] "edge_push" builder);
           expr builder n1;
     | SPushBack(s, e) -> (match (s, e) with
          ((A.List(t), l), _) -> let e' = expr builder e in
          (match t with
             A.Int | A.Float ->
       let ptr = L.build_malloc (ltype_of_typ t) "element" builder in
       ignore (L.build_store e' ptr builder);
       let cast = L.build_bitcast ptr void_ptr_t "cast" builder in
         ignore (L.build_call list_push_back_f
         [| expr builder s; cast|] "list_push_back" builder);
         expr builder (A.List(t),l)
           | _ -> let cast = L.build_bitcast e' void_ptr_t "cast"
builder in
         ignore (L.build_call list_push_back_f
         [| expr builder s; cast|] "list_push_back" builder);
         expr builder s)

         | _ -> raise (Failure ("internal error on pushback"))  )
     | SPushFront(s, e) -> (match (s, e) with
          ((A.List(t), l), _) -> let e' = expr builder e in
          (match t with
             A.Int | A.Float ->
       let ptr = L.build_malloc (ltype_of_typ t) "element" builder in
       ignore (L.build_store e' ptr builder);
       let cast = L.build_bitcast ptr void_ptr_t "cast" builder in
         ignore (L.build_call list_push_front_f
         [| expr builder s; cast|] "list_push_front" builder);
         expr builder (A.List(t),l)
           | _ -> let cast = L.build_bitcast e' void_ptr_t "cast"
builder in
         ignore (L.build_call list_push_front_f
         [| expr builder s; cast|] "list_push_front" builder);
         expr builder s)

         | _ -> raise (Failure ("internal error on pushback"))  )
     | SPopBack(s) -> (match s with
           (A.List(t), _)  ->
          (match t with
             A.Int | A.Float ->
             (let ptr = (L.build_call list_pop_back_f
           [| expr builder s|] "list_pop_back" builder)
           and ty = (L.pointer_type (ltype_of_typ t)) in
           let cast = L.build_bitcast ptr ty "cast" builder in
           L.build_load cast "val" builder)

           | _ ->  (let ptr = (L.build_call list_pop_back_f
           [| expr builder s |] "list_pop_back" builder)
           and ty = (ltype_of_typ t) in
           L.build_bitcast ptr ty "cast" builder)
           )
         | _ -> raise (Failure "internal error on pop_back") )
     | SPopFront(s) -> (match s with
           (A.List(t), _)  ->
          (match t with
             A.Int | A.Float ->
             (let ptr = (L.build_call list_pop_front_f
           [| expr builder s|] "list_pop_front" builder)
           and ty = (L.pointer_type (ltype_of_typ t)) in
           let cast = L.build_bitcast ptr ty "cast" builder in
           L.build_load cast "val" builder)

           | _ ->  (let ptr = (L.build_call list_pop_front_f
           [| expr builder s |] "list_pop_front" builder)
           and ty = (ltype_of_typ t) in
           L.build_bitcast ptr ty "cast" builder)
           )
         | _ -> raise (Failure "internal error on pop_back") )
     | SPeekBack(s) -> (match s with
           (A.List(t), _)  ->
          (match t with
             A.Int | A.Float  ->
             (let ptr = (L.build_call list_peek_back_f
           [| expr builder s|] "list_peek_back" builder)
           and ty = (L.pointer_type (ltype_of_typ t)) in
           let cast = L.build_bitcast ptr ty "cast" builder in
           L.build_load cast "val" builder)

           | _ ->  (let ptr = (L.build_call list_peek_back_f
           [| expr builder s |] "list_peek_back" builder)
           and ty = (ltype_of_typ t) in
           L.build_bitcast ptr ty "cast" builder)
           )
         | _ -> raise (Failure "internal error on peek_front") )
     | SPeekFront(s) -> (match s with
           (A.List(t), _)  ->
          (match t with
             A.Int | A.Float ->
             (let ptr = (L.build_call list_peek_front_f
           [| expr builder s|] "list_peek_front" builder)
           and ty = (L.pointer_type (ltype_of_typ t)) in
           let cast = L.build_bitcast ptr ty "cast" builder in
           L.build_load cast "val" builder)

           | _ ->  (let ptr = (L.build_call list_peek_front_f
           [| expr builder s |] "list_peek_front" builder)
           and ty = (ltype_of_typ t) in
           L.build_bitcast ptr ty "cast" builder)
           )
         | _ -> raise (Failure "internal error on peek_front") )
     | SAddNode(g, e) ->  L.build_call graph_add_node_f
         [| expr builder g; expr builder e|] "graph_add_node" builder
     | SGAdd(g, id, v) -> (match g with
         (A.Graph(t), _) ->
           let nptr = L.build_malloc (ltype_of_typ t) "nodeval"
builder in
           ignore(L.build_store (expr builder v) nptr builder);
           let cast = L.build_bitcast nptr void_ptr_t "cast" builder
in
```

```
            L.build_call graph_add_f
            [| expr builder g; expr builder id; cast |] "graph_add"
builder
          | _ -> raise (Failure "internal error on add"))
        | SContains(g, n) -> L.build_call graph_contains_node_f
            [|expr builder g; expr builder n|] "graph_contains_node"
builder
        | SContainsId(g, id) -> L.build_call graph_contains_id_f
            [|expr builder g; expr builder id|] "graph_contains_id"
builder
        | SAddAll(s, el) -> let s' = expr builder s in (match s with
            (A.List(t),_) -> (match t with
                A.Int | A.Float -> let addfun e = (
            let e' = expr builder e in
            let ptr = L.build_malloc (ltype_of_typ t) "element" builder
in
            ignore (L.build_store e' ptr builder);
            let cast = L.build_bitcast ptr void_ptr_t "cast" builder in
            ignore(L.build_call list_push_back_f
            [| s'; cast|] "list_push_back" builder)) in
            List.iter addfun el;
            s'
              | _ ->  let addfun e = (
            let e' = expr builder e in
            let cast = L.build_bitcast e' void_ptr_t "cast" builder in
            ignore(L.build_call list_push_back_f
            [| s'; cast|] "list_push_back" builder)) in
            List.iter addfun el;
            s')
          | (A.Graph(t), _) -> let l = expr builder
            (A.List(A.Node(t)), SAccess(s, "edges"))
          in (match t with
                A.Int | A.Float -> let addfun e = (
            let e' = expr builder e in
            let ptr = L.build_malloc (ltype_of_typ t) "element" builder
in
            ignore (L.build_store e' ptr builder);
            let cast = L.build_bitcast ptr void_ptr_t "cast" builder in
            ignore(L.build_call list_push_back_f
            [| l; cast|] "list_push_back" builder)) in
            List.iter addfun el;
            s'
              | _ ->  let addfun e = (
            let e' = expr builder e in
            let cast = L.build_bitcast e' void_ptr_t "cast" builder in
            ignore(L.build_call list_push_back_f
            [| l; cast|] "list_push_back" builder)) in
            List.iter addfun el;
            s')
          | _ -> raise (Failure "internal error on add_all"))
        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 = let predval = expr builder predicate in
              (match predicate with
                (A.Int, SBinop(_)) -> predval
              | (A.Int, _) ->
             L.build_icmp L.Icmp.Sgt predval (L.const_int i32_t 0) "tmp"
   builder
              | (A.Float, _) ->
            L.build_fcmp L.Fcmp.Ogt predval (L.const_float float_t 0.0)
                "tmp" builder
              | _ -> raise (Failure "internal error on if-pred")) 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 = let predval = expr pred_builder predicate in
              (match predicate with
                (A.Int, SBinop(_)) -> predval
              | (A.Int, _) ->
             L.build_icmp L.Icmp.Sgt predval (L.const_int i32_t 0)
              "tmp" pred_builder
              | (A.Float, _) ->
            L.build_fcmp L.Fcmp.Ogt predval (L.const_float float_t 0.0)
                "tmp" pred_builder
              | _ -> raise (Failure "internal error on if-pred")) 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]) ] )
       | SDeclare(A.List(_), ll, _) -> List.iter (fun s ->
           let ptr = L.build_call list_init_f [| |] "init_list"
builder in
           ignore (L.build_store ptr (lookup s) builder)) ll; builder
       | SDeclare(A.Graph(_), lg, _) -> List.iter (fun g ->
           let ptr = L.build_call graph_init_f [| |] "init_graph"
builder in
           ignore (L.build_store ptr (lookup g) builder)) lg ;
builder
       | SDeclare(A.Node(t), ln, _) -> List.iter (fun n ->
           let ptr = L.build_call node_init_f [| |] "init_node"
builder in
           ignore (L.build_store ptr (lookup n) builder)  ;
           let val_ptr =
             L.build_struct_gep ptr 1 "struct.ptr" builder in
           let mptr = L.build_malloc (ltype_of_typ t) "nodeval"
builder in
            let cast = L.build_bitcast mptr void_ptr_t "cast" builder
in
           ignore (L.build_store cast val_ptr builder)) ln ; builder
       | SDeclare(_, _, a) -> ignore(expr builder a); builder

     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
```

8.7 graphene.ml

```
(*
    Graphene Compiler
    Author: Ashar Nadeem
*)

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: ./graphene.native [-a|-s|-l|-c] [file.gph]"
in
 let channel = ref stdin in
 Arg.parse speclist (fun filename -> channel := open_in filename)
usage_msg;
  let lexbuf = Lexing.from_channel !channel in
 let ast = Parser.program Scanner.token lexbuf in
 match !action with
   Ast -> print_string (Ast.string_of_program ast)
 | _ -> let sast = Semant.check ast in
   match !action with
     Ast     -> ()
   | Sast    -> print_string (Sast.string_of_sprogram sast)
   | LLVM_IR -> print_string (Llvm.string_of_llmodule
(Codegen.translate sast))
   | Compile -> let m = Codegen.translate sast in
 Llvm_analysis.assert_valid_module m;
 print_string (Llvm.string_of_llmodule m)
```

8.8 graphene.h

```
/*
   header file for Graphene built-in functionality
   Author: Ashar Nadeem and Matthew Sanchez
*/

#prama once


struct list_element
{
   void *element;
   struct list_element *next;
};

struct list
{
   int size;
   struct list_element *head;
};

struct edge
{
   void *weight;
   struct node *dest;
   int t;
};

struct node
{
   int id;
   void *val;
   struct list *edges;
};
```

```
struct graph
{
   struct list *nodes;
   struct node *root;
};

struct list *list_init();
void *list_index(struct list *list, int index);
void list_push_back(struct list *list, void *element);
void list_push_front(struct list *list, void *element);
void *list_pop_back(struct list *list);
void *list_pop_front(struct list *list);
void *list_peek_back(struct list *list);
void *list_peek_front(struct list *list);

struct node *node_init();
void node_set_id(struct node *node, int id);
void node_set_val(struct node *node, void *val);

struct edge *edge_init(void *w, struct node *n, int tr);

struct graph *graph_init();
int graph_size(struct graph *graph);
void graph_add_node(struct graph *graph, struct node
*node);
struct node *graph_add(struct graph *graph, int id, void
*val);
struct node *graph_get_node(struct graph *graph, int key);
int graph_contains_node(struct graph *graph, struct node
*node);
int graph_contains_id(struct graph *graph, int id);

int string_cmp(void *v1, void*v2);
```

8.9 graphene.c

```
/*
   functions implementing Graphene build-in functionality
   Author: Ashar Nadeem & Matthew Sanchez
*/

#include <stdlib.h>
#include "graphene.h"

/* ---------- List Functions ---------- */

struct list *list_init()
{
   struct list *list = malloc(sizeof(struct list));
   list->size = 0;
   list->head = NULL;
   return list;
}

void *list_index(struct list *list, int index)
{
   int cur = 0;

   struct list_element *tmp = list->head;
   while (tmp != NULL)
   {
       if (cur == index)
       {
           return tmp->element;
       }
       else
       {
           tmp = tmp->next;
           cur++;
       }
   }
   return 0;
}

void list_push_back(struct list *list, void *element)
{
   struct list_element *node = (struct list_element
*)malloc(sizeof(struct list_element));

   node->element = element;
   node->next = NULL;

   if (list->head == NULL)
   {
       list->head = node;
       list->size = list->size + 1;
       return;
   }

   struct list_element *tmp = list->head;
   while (tmp->next != NULL)
   {
       tmp = tmp->next;
   }

   tmp->next = node;
   list->size = list->size + 1;
}

void list_push_front(struct list *list, void *element)
{
   struct list_element *node = (struct list_element
*)malloc(sizeof(struct list_element));
   node->element = element;
   node->next = (list->head);
   list->head = node;
   list->size = list->size + 1;
}

void *list_pop_back(struct list *list)
{
   struct list_element *tmp = list->head;
   struct list_element *t;
   if (list->head->next == NULL)
   {
       void *data = list->head->element;
       free(list->head);
       list->head = NULL;
       list->size = list->size - 1;
       return data;
   }
   else
   {
       while (tmp->next != NULL)
       {
           t = tmp;
           tmp = tmp->next;
       }
       void *data = t->next->element;
       free(t->next);
       t->next = NULL;
       list->size = list->size - 1;
       return data;
   }
}

void *list_pop_front(struct list *list)
{
   if (list->head != NULL)
   {
       void *data = list->head->element;
       struct list_element *tmp = list->head->next;
```

```
        free(list->head);
        list->head = tmp;
        list->size = list->size - 1;
        return data;
    }
    return NULL;
}

void *list_peek_back(struct list *list)
{
    struct list_element *tmp = list->head;
    while (tmp->next != NULL)
    {
        tmp = tmp->next;
    }
    return tmp->element;
}
void *list_peek_front(struct list *list)
{
    return list->head->element;
}


/* ---------- End List Functions ---------- */

/* ---------- Node Functions ---------- */

struct node *node_init()
{
    struct node *node = malloc(sizeof(struct node));
    //node->val = malloc(sizeof(void *));
    node->edges = malloc(sizeof(struct list));
    return node;
}

/* ---------- End Node Functions ---------- */
/* ---------- Edge Functions ----------*/

struct edge *edge_init(void *w, struct node *n, int tr)
{
   struct edge *edge = malloc(sizeof(struct edge));
   edge->weight = w;
   edge->dest = n;
   edge->t = tr;
   return edge;
}

void edge_check(struct node *n1, struct node *n2)
{
}
/* ---------- End Edge Functions ---------- */

/* ---------- Graph Functions ---------- */

struct graph *graph_init()
{
   struct graph *graph = malloc(sizeof(struct graph));
   graph->nodes = list_init();
   graph->root = NULL;
   return graph;
}

void graph_add_node(struct graph *graph, struct node
*node)
{
   if (graph->root == NULL)
   {
       graph->root = node;
   }

   struct list_element *tmp;
   tmp = graph->nodes->head;
   if (tmp == NULL)
   {
     struct list_element* newel = (struct list_element*)
malloc(sizeof(struct list_element));
     newel->element = node;
     newel->next = NULL;
     graph->nodes->head = newel;
   }
   else {
     while (tmp->next != NULL)
     {
         tmp = tmp->next;
     }

     struct list_element* newel = (struct list_element*)
malloc(sizeof(struct list_element));
     newel->element = (void *) node;
     newel->next = NULL;
     tmp->next = newel;


   }
   graph->nodes->size = graph->nodes->size + 1;
}

struct node *graph_add(struct graph *graph, int id, void
*val)
{
   struct node *node = node_init();
   node->id = id;
   node->val = val;
   graph_add_node(graph, node);
   return node;
}
```

```c
struct node *graph_get_node(struct graph *graph, int id)
{
   struct list_element *tmp;
   tmp = graph->nodes->head;

   while (tmp != NULL)
   {
       if(((struct node *) tmp->element)->id == id)
       {
           return (struct node *) tmp->element;
       }
       tmp = tmp->next;
   }
   return NULL;
}
int graph_contains_node(struct graph *graph, struct node
*node)
{
   struct list_element *tmp = graph->nodes->head;
   while (tmp != NULL)
   {
       if (tmp->element == node)
       {
           return 1;
       }
       tmp = tmp->next;
   }
   return 0;
}

int graph_contains_id(struct graph *graph, int id)
{
   return graph_get_node(graph, id) != NULL;
}

/* ---------- End Graph Functions ---------- */
/* ---------- Misc. Functions ---------- */

int string_cmp(void *v1, void *v2)
{
   char *s1 = (char *)v1;
   char *s2 = (char *)v2;
   while (*s1 && *s2)
   {
       if (*s1++ == *s2++)
       {
       }
       else
       {
           return 0;
       }
   }
   if (*s1 == *s2)
   {
       return 1;
   }
   else
   {
       return 0;
   }
}
```