Grape Language Reference Manual

2018 Fall Programming Languages and Translators

James Kolsby	*jrk2181*
Nick Krasnoff	*nsk2144*
HyunBin Yoo	*hy2506*
Wu Po Yu	*pw2440*

December 19, 2018

1. Introduction

Graph algorithms are an extremely ripe domain for networks and relationships of data. Graph algorithms can be very useful in a wide range of applications, including databases, network flow problems, and even language parsing using finite automata. Grape is a language that is designed to make the assembly and manipulation of graphs much more visually comprehensible and syntactically convenient. It should allow its user to implement programs like DFA simulation or Dijkstra's algorithm easily and concisely.

2. Data Types

2.1. Primitive Types

Int - A 32-bit signed integer designated by a series of digits

Bool - A 1-bit boolean designated by ***Yes*** or ***No***

Float - An signed double precision flowing number designated by a sign, a decimal and an exponent.

String - A series of characters that are enclosed in double quotes

2.1.1. Examples of Primitives

```
Int i = 1;
Bool b = No;
Float pi = 3.14;
String me = "James";
```

2.2. Reference Types

List - A collection which is ordered and mutable. It is designated by a series comma-delimited expressions enclosed in square brackets, like so:

```
List<Int> a = [1, 2, 3];
List<String> b = ["Hello", "World"];
```

2.3. Graph Types

Node - A node is a container representing a vertex in a graph, designated by an expression enclosed in single quotes, like so:

```
Node<Int> a = '3';
Node<String> b = '"Hello"';
```

Edge - An edge is an object that represents a directed relationship between two nodes, designated by an expression enclosed in hyphens with a closing bracket representing its directionality. As with a node, the expression contained in an edge can be of any type, for instance an integer containing a cost of traversing that edge. For instance: ***Edge<Int> a = << -3- >>;***

An edge contains references to two nodes, a source and a destination. These nodes can be accessed as properties of the edge.

```
Edge<Int> c;
...
c.to;
c.from;
```

Graph - A graph is a collection of Nodes and Edges that can be interconnected or disjoint. Graph initialization is designated by a space-delimited path of nodes and edges, enclosed in double angle brackets, like so:

```
Graph<String, Int> x = <<'"Atlanta"' -4- '"New York"'>>;
```

Graphs can contain any Nodes of any type, and can mix types. More complicated graphs can be described using a comma-delimited series of paths. Reference names can be passed into the graph.

```
Node<String> a = '"Atlanta"';
Graph<String, Int> cities = <<a -5- '"Charleston"', a -30-
    '"New York"', a -100- '"San Francisco"'>>;
```

Graph initialization can be used to describe paths wherein two edges share a common node between them, for instance:

```
Graph<Int, Int> path = <<'1' -30- '2' -40-> '3'>>
```

These paths are evaluated from left to right, where the ***from*** of each subsequent Edge is the same as the ***to*** of the Edge preceding it. In the above example, the Nodes containing Integers 1 and 3 are both connected to the node containing '2' via the Edges containing 30 and 40 respectively.

3. Operators and Expressions

3.1. Variable Declaration

A variable must be declared in the top-level scope of a function, and must be provided with a type. A declaration can also contain an assignment, however it is optional.

3.2. Variable Assignment

The `=` operator is used for a variable assignment. The right-hand expression is evaluated and its value is assigned to the left-hand typed ID. LHS and RHS must have the same type. This operator will be evaluated right-to-left.

3.3. Arithmetic Operators

The arithmetic operators are `*` (Multiplication), `/` (Division), `+` (Addition) and `-` (Subtraction). They are all binomial operators. The minus sign can also be used as a unary operation to invert a number's sign (Negation).

3.4. Relational Operators

The relational operators are `<` (Less than) `>` (Greater than) `<=` (Less than or equal to) `=>` (Greater than or equal to) `==` (Equal to) `!=` (Not equal to). They are evaluated from left to right. They each require two values which are to be compared and will return ***Yes*** if the comparison is truthful and ***No*** otherwise.

3.5. Boolean Operators

The logical operators are ***not***, ***and***, and ***or***. Not negates the subsequent boolean, while and and or both return the logical comparison of the values on either side of them, like so:

```
Bool t = Yes
Bool f = No
Bool yes = t or f
Bool no = not t
```

3.6. List Indexing

An object of List or String type can be indexed using bracket notation. A list index will return the type of the objects contained in the list, and a string index will return a string type with unit length.

```
List<Int> a = [1,2,3,4];
String b = "hello"
print(a[2]);   // Prints 3
print(b[2]);   // Prints l
```

3.7. Value Property

Nodes and edges are internally represented using structs. To access the properties of these structs, one can use dot notation.

```
Node<Int> a = '5';
print(a.val);  // Prints 5
```

3.8. List Operations

list_get (Int x, List<Node<Int> > y) - Get a Node at index x from y

push_front_list_node(Node<Int> x, List<Node<Int> > y) - Push x to the front of y, return Void

update_at (Int x, List<Node<Int> > y, Node<Int> z) - Update the element in y at index x with z, return the updated list.

3.9. Graph Operations

graph_to_list(Graph<Int,Int>) - Return a list of Node<Int> from x

neighbor(Node<Int>) - Return all neighbors of x in a list of Node<Int>

distance(Node<Int>, Node<Int>) - Return an Int distance between node x, y.

dfa(Graph<Bool,String>, Node<Bool> start, String input) - Simulate a DFA on a Graph of Booleans where the accepting states contain Yes and the edges are input tokens.

dist(Graph<Int,Int>, Node<Int> start, Node<Int> end) - Compute Dijstra's Algorithm between two Nodes to find the shortest path between them.

3.10. Node Operations

node_same(Node<Int> x , Node<Int> y) - compare two Node<Int>, determine if they share the same pointer

update_node(Int x , Node<Int> y) - update y's Int value to x, return the updated node

3.11. Precedence and Order of Operations

Parentheses have the highest priority in the evaluation of expressions. Logical and relational operators have lower precedence than the arithmetic operators, so statements including that include logical or relational operators alongside arithmetic operations will evaluate the arithmetic statement first and then apply relational and logical operators to them in that order. For instance this statement evaluates to ***Yes***:

```
Bool yes = 3 > 5 - 2 and (2 + 2 <= 4)
```

4. Programming Structure

Grape programs are described as a single source file which contains a series of global statements or function declarations which are evaluated from top to bottom.

4.1. Blocks and Statements

Grape is an imperative programming language and is designed to be written in blocks, a series of statements which are executed top to bottom. Statements within a block are delimited by semicolons, and can span an arbitrary number of lines.

4.2. Comments

Single-line comments are designated by a double forward slash, and are terminated by a new line. Multi-line comments are designated by three forward slashes, and are terminated by another three forward slashes.

```
Int a = 5; // Look ma a comment!

///
      Welcome to the COMMENT ZONE!
///
```

4.3. Functions

Functions act as a way to compartmentalize segments of your program. Functions are defined by a return type, an ID, and zero or more comma-delimited parameters enclosed in parentheses. The function body consists of a series of statements that must contain a return statement specifying the value to be returned. A function declaration is designated as follows:

```
fun Int mod(Int a, Int b) {
        Int m;
        Int k = 0;
        Int left = 0;
        Int right = a;
        while ( left < right ) {
            m = (left + right) / 2;
            if ( a - m*b >= b ) {
               left = m + 1;
            } else {
               right = m;
            }
        }
        return a - left * b;
}
```

```
fun Int gcd(Int a, Int b) {
    Int c;
    while (b > 0) {
        Int c = mod(a,b);
        a = b;
        b = c;
    }
    return a;
}

// Don't forget main()
fun Int main() {
    print(gcd(1234,321));
}
```

Functions can be called in any function body. A function call is designated by the function ID and a series of parameters enclosed in parentheses:

```
Int a = gcd(10, 15);
```

5. Control Flow

5.1. Conditionals

Grape supports if statements that may contain an optional else condition to execute if the given condition is false.

```
if (r == 0) { return 3; }
else { return 2; }
```

5.2. Loops

while loops are designated by a looping condition and a block to be executed as long as the condition is truthful. They are designated as follows:

```
while (x < 5) {
     x = x + 1;
}
```

6. Standard Library

The Grape standard library provides useful built-in methods for manipulating the List, Dict, and Graph types, as well as for standard I/O:

print(a) - writes a to stdout.
size(a) - returns the Int size of a List or String.

6.1. List Methods

l.get_node(i) – returns a data of the ith node
l.insert(i,x) - insert variable x at index i and return the modified List
l.pop_list() - remove and return the last element from the List
l.pop_list_front() - remove the element from the beginning of the List and return the last element from the List
l.push_list(x) - add the element to the end of the List and return the last element from the List
l.push_list_front(x) - add the element to the beginning of the List and return the first element from the List
l.remove(x) - remove the first instance of x and return the modified List

6.2. Dict Methods

d.size() - return the number of keys in the Dict

d.key(x) - return the key of the value x, if it exists in the Dict.

d.remove(x) - remove the key x and its value from the Dict

6.3. Graph Methods

g.Graphsize() - return the number of Nodes in the Graph

g.GraphLeaves() - return a list of Nodes with only one incoming Edge

g.neighbor(x) - return a List of Nodes adjacent to Node x

g.GraphFind(x) - return the list of all Nodes that contain the value x

g.GraphIsEmpty() - return True of the Graph is empty, otherwise return False

g.distance(x, y) - return distance between Node x and y

g.graph_to_list() – get all the nodes in the graph and return a list containing all the nodes

g.outgoing(node) – get all edges that are outgoing from the node

g.addnode(a) – add node a to the graph

g.addedge(e) – add edge e to the graph

7. Project Plan

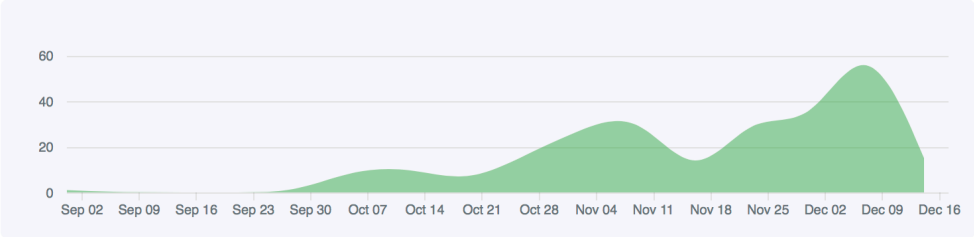
7.1. Weekly Meet-ups:

In order for the group to slowly establish the codebase, we scheduled a meeting once a week with our TA, Dean Deng. We also met at least two more times a week to discuss the project goals. At the start, these meeting were primarily discussions of broader concept, of the syntax, and the functionality of the language and throughout the semester, the team incrementally met more and more often, shifting from discussion to delegating technical tasks to each individual.

7.2. Workflow:

All of the project is located within Docker, because it allows us to streamline to development among teammates. However, while programming, members can either choose work on local repository that is mounted onto the docker or the docker container itself. As we make changes throughout the project, we uses Git version control and Github, and at times, worked from new branches to test new features. To ensure that there weren't erroneous code being pushed onto the shared repository, every member can only push compilable code.

The graph below plots the project's commits over the course of the semester. As one can tell, there are incrementally more and more commits at the second half of the semester, as we finalize the language and make necessary changes for



7.3. Programming Style Guide:

- Always use tabs when we need to indent.
- For the C library, we strive to make reasonable function names to ensure clarity
- At the start of a git commit message, denotes whether the commit is a “feat” or “fix”.
- Comment on particularly intricate code to convey meaning clearly.

7.4. Language Evolution

This LRM deviates from the original proposal in a number of ways due to time restraints. The largest feature that we had to cut was graph templating, which would have made searching graphs much easier. Instead, we opted to build out more general features which could make writing graph algorithms more syntactically convenient. For instance, we added list and string indexing using brackets, object properties, and methods to easily pass data around inside graphs.

7.5. Project timeline:

During the semester, we scheduled milestones to achieve throughout the semester and strived to actualize them. In reality, the process turned out to be more of a back-and-forth trajectory as a lot of times, we realize that there are previous mistakes or designs that we wanted to alter.

7.6. Milestones

Date	Milestone
10/15	LRM, Scanner done, elementary Parser
11/18	Parser, AST, SAST, "Hello World"
11/25	Semantically checked types (edges, nodes)
12/2	Edge, Node, List typing in codegen.ml
12/10	Graph type in codegen.ml
12/11	Writing C library, Linking C library
12/12	List indexing, Dot notation, Overloading functions

8. Roles and Responsibility:

8.1. Language Guru - James Kolsby

The language guru, a true visionary, dictates the utility and aesthetics of the language, Grape. As he caressed the purple marbles bursting of nature's nectar, he gasped in Archimedean fashion: "A grape cluster is basically a tree, a graph!"

8.2. Project Manager - Po Yu (Timmy) Wu

The project manager primary deals with humans, the only part in the project that are, unfortunately, not made of booleans. He strives to keep the project afloat by keeping track of the schedule, coaxing the hearts and minds of self-absorbed Columbia students, and, oftentimes, making announcements in a messaging group that sometimes seems to be constituted of only he, himself and "hi".

8.3. System Architect - HyunBin (Edward) Yoo

The system architect decides the software development environment that the group operates in and just like Meryl Streep in the Sophie's choice, the architect knows all too well that no matter what he chooses, he and his team would carry the burden, forever haunted.

8.4. Tester - Nick Krasnoff

The tester ensures that the code that we so diligently farmed truly comes to fruition and works the way we wanted it to. His responsibility includes building test suites, writing test scripts, and having inhumanely amount of patience for his trial and error was full of disappointments and, well, errors.

9. Lessons Learned

9.1. James

I think one of the most important things that this project taught me was how the compiler works at each individual level. This project which seemed completely impossible at the beginning proved to be just a series of small transformations to the input. Codegen was the most mystifying piece of the project, but just turned out to be a traversal of the SAST using the LLVM builder to assemble each block required by the program. I am really happy to have learned about the LLVM infrastructure and intermediate representations in general. During code generation, the countless segmentation faults that we encountered proved the importance of semantic analysis for proper casting of different types. Using the types that were statically computed during the semantic stage of compilation, it is very easy to overload operations to handle any data size or type. There were several functions that I worked on overloading, such as indexing lists and strings or finding the size of a list or string. Though these were completely different functions in our C backend, it was easy to get the type and cast the function return. I made an unsuccessful attempt at permitting polymorphism in certain functions by having each declaration store its return type as a function of its argument types. This would be especially helpful for graph methods which will return Nodes and Edges of different types

depending on the type of the Graph. We managed to overload print and size only because they always return void and integer respectively.

9.2. Timmy (Po Yu)

With this project, I have picked up a few technical skills and also learned to work in a team-working setting. Other than the course materials, I found the group working aspect of the project particularly enlightening. Throughout the process, each team members worked in distinct ways and learned the material in different pace. As the project manager, I strived to keep everyone on board, on-time for the deliverables in order to establish a well formed codebase steadily. However, going into the project, I found that there are a natural group tendency to resolve to inertia, especially with everyone occupied with their upcoming school works. Inevitably, a project that is due at the end of the semester could be put on lower priority. To break away from such group dynamic, I learned that it is sometime best if the project manager took on a more decisive role, making choices when there is an impasse or no response at all. This, of course, is not to suggest that each group member's opinion should not be heard, but that when there is no firm meeting date settled down, no particular, urgent objectives to achieve in the immediate future, the project manager should nonetheless, pin down a meeting time, a technical issue that the group should address in order to move forward smoothly.

9.3. Nick

With this project, I have picked up a few technical skills and also learned to work in a team-working setting. Other than the course materials, I found the group working aspect of the project particularly enlightening. Throughout the process, each team members worked in distinct ways and learned the material in different pace. As the project manager, I strived to keep everyone on board, on-time for the deliverables in order to establish a well formed codebase steadily. However, going into the project, I found that there are a natural group tendency to resolve to inertia, especially with everyone occupied with their upcoming school works. Inevitably, a project that is due at the end of the

semester could be put on lower priority. To break away from such group dynamic, I learned that it is sometime best if the project manager took on a more decisive role, making choices when there is an impasse or no response at all. This, of course, is not to suggest that each group member's opinion should not be heard, but that when there is no firm meeting date settled down, no particular, urgent objectives to achieve in the immediate future, the project manager should nonetheless, pin down a meeting time, a technical issue that the group should address in order to move forward smoothly.

9.4. Edward (HyuBin)

I learned several lessons while working on this semester-long project. First of all, the communication among team members is very important. As the semester passed, each of us were taking different classes and has different schedule. It was very important to communicate with each other on assigning the tasks while respecting each other's schedule. I would definitely say that the communication between team members for this kind of long project is the key to the success. In addition, it was my first encounter to the functional programming language and it was not easy to pick up. I would say that I am still not very comfortable with the functional language, but it was a really great chance to encounter different type of language other than Java, C, C++, Python, and etc. Learning Ocaml, a functional language, was not easy, but I can definitely understand why people are using a functional languages to program. It definitely has some features that cannot be implemented by using other languages. Lastly, I was very glad that I was able to apply the concepts that we learned in other CS classes such as Computer Science Theory, Fundamentals of Computer Systems, and Advanced Programming. When I was taking those classes, for example, CS Theory, I was not sure where I will be using DFA, NFA, and other concepts that we learned in the class. This project was actually the first chance to use those concepts to build something. I would say this chance of being able to apply the theoretical concepts of Computer Science was the most valuable experience that I had while working on the project.

10. Architectural Design

Grape's compiler is constituted of several files, with functionalities of their own. There is a single ***src/*** folder that holds ***scanner.mll***, ***parser.mly***, ***ast.ml***, ***sast.ml***, ***semant.ml***, ***codegen.ml***, a standard library written in C which is linked within codegen, and a standard library written in Grape which is linked at compile time. The ***dist/*** folder holds the compiled object, along with the ***grplib.grp*** to be concatenated with a program before it is compiled, and the ***stdlib.o***, which is linked with the resulting object file using ld. This ensures the definition of our standard C functions. All tests are kept in ***tests/***, along with a ***.out*** file which stores the program's output, a ***.ast.out*** file which stores the program's AST, and a ***.sast.out*** which stores the semantic AST. Below are the contributions to each component:

Scanner	James Kolsby, Nick Krasnoff, Po Yu (Timmy) Wu
Parser	Edward Yoo, James Kolsby, Nick Krasnoff, Po Yu (Timmy) Wu
Semant and Codegen	James Kolsby, Nick Krasnoff, Po Yu (Timmy) Wu
C Library	Edward Yoo, Nick Krasnoff, Po Yu (Timmy) Wu, Edward Yoo

11. Testing and Running Grape

The DFA program is a useful program that allows a user to input a DFA in the form of a graph, the start node, as well as an input string to determine if the input is accepted by the DFA. Dijkstra's algorithm is another solid Grape Program that allows a user to find the shortest path between two nodes in a graph.

In order to test the Grape language we created test files to isolate the different components of Grape which allowed us to quickly identify the source broken features as

well as make sure existing capabilities were not broken whenever new features were added.

Our testing suite functions by running a script called test which runs through the tests/ directory compiles our grp source file test cases and generates the ast, sast, and output for each test case and compares them to their corresponding .out files whenever the script is ran. These .out files are generated if the test script cannot find those files, thus it was important for us to make sure the .out files were accurate upon creation as when we modify a test case we must remove the output files and new ones will automatically be created. Additionally, whenever a test did fail a .fail file would be generated that contains the failing output.

We chose our test cases to be the most fundamental operations of our language such that every program in our language would ideally be made up of only things our test case covered. For example we have test files to test our data types such as list.grp, node.grp, and graph.grp as well as testing how they interact with each other such as in multi-list.grp. We also have some cases such list-out-of-bounds.grp that test how Grape reacts to invalid inputs. Additionally, whenever we ran into a problem that our existing test cases didn't cover we would create additional test cases to try to replicate the error. Due to how we created our test cases we could quickly narrow down the source of an error.

The automation we used to develop and test Grape entails of Bash scripts to compile our programs and set up our docker environments. We also used Makefiles to compile our grape.native file and C libraries.

Script Usage:

`1. Environment Setup`

2. In the top level directory

   2.1. Initialize docker development environment: ./run init

   2.2. Reopen docker after initial creation: ./run dev

Compilation

To compile the translator move to the src directory and run make clean followed by make. This creates the dist directory in the top level containing the c library object file(stdlib.o), a compilation script(grape), the grape.native file, and the grape standard library(grplib.grp).

Proceeding from there,

```
./run test
```

At the top level would run the test suite that would print out fail or pass statements in the bash terminal. If one desires to test individual files, then one can go into the dist directory, and

```
./grape ../tests/<target.grp>
```

If <target.grp> is a parsable program in Grape, this should output an executable ./out.g file in the same directory. From there, running

```
./out.g
```

would execute the <target.grp> file.

12. Development Tools and Languages:

Need	Tools
Communication	SMS, Slack
Version Control	Git, Github
Editor	Vim, Docker, Sublime Text, Merlin
Languages	Ocaml, C Language, Bash, Yacc, Lex
Documentation	Github Markdown, Google Docs,

13. Advice for Future PLT Students

13.1. Everyone has a different learning/ working pace. Make sure that you are respecting other's.

13.2. Be self-aware of what and how you can contribute to the team

13.3. Make sure that you make space for people in the group who aren't so quick to share their ideas

13.4. You can do it!! It will all fall into place one day.

14. Project Log

```
f271103 2018-12-19 James Kolsby feat: presentation tests (HEAD ->
   presentation)
75cc06f 2018-12-19 James Kolsby feat: hiBob test
3ed5a42 2018-12-19 James Kolsby fix: function args
d389d2c 2018-12-19 Nick Krasnoff test scripts remove fail before run and
   not after (origin/presentation)
570074d 2018-12-19 Timmypoyu editted out the comment in c library
5986ba4 2018-12-19 Nick Krasnoff Working DIjkstra
6d28706 2018-12-19 Nick Krasnoff Merge branch 'presentation' of https://
   github.com/jrkolsby/Grape into presentation
b949e4f 2018-12-19 Nick Krasnoff Dijkstra tinkering
53618f7 2018-12-19 James Kolsby fix: args were reversed...
227b857 2018-12-18 James Kolsby feat: added new LRM
26b7680 2018-12-18 Nick Krasnoff Cooler dfa algorithm
62a3d1b 2018-12-18 Nick Krasnoff tests and no dup nodes
```

```
ccf0ccd 2018-12-18 Nick Krasnoff Missing distance
ee75832 2018-12-18 Tim added graph functions
61004c5 2018-12-18 Nick Krasnoff feat: Correctly reordered SCALL
9447015 2018-12-18 Nick Krasnoff Dijkstra.grp now has: Fatal error:
   exception Failure('unrecognized function update_node')
c89cc74 2018-12-18 Tim debugging dijkstra
264c034 2018-12-18 Tim added: some functions needed for dijkstra
406bfd0 2018-12-17 Tim feat:add node_same
9529f28 2018-12-18 root add function to semant and change dij(not done)
3e81a84 2018-12-17 Nick Krasnoff Merge branch 'presentation' of https://
   github.com/jrkolsby/Grape into presentation
2e86fec 2018-12-17 Nick Krasnoff modified test script and added Dijkstra
ff17f47 2018-12-17 James Kolsby fix: grplib
2efba40 2018-12-17 James Kolsby fix: update tests
6c264f3 2018-12-17 James Kolsby fix: tests
4fbe253 2018-12-17 James Kolsby fix: func_decl keys
af8e1cc 2018-12-17 James Kolsby feat: overloading size and print
8386915 2018-12-16 James Kolsby feat: fixed test cases (develop)
ab1537d 2018-12-15 James Kolsby fix: assign semantics
a2211a7 2018-12-15 Nick Krasnoff Merge branch 'develop' of https://
   github.com/jrkolsby/Grape into develop
96452a4 2018-12-15 Nick Krasnoff additional test cases
2235b91 2018-12-15 James Kolsby fix: dfa
c91f590 2018-12-15 Nick Krasnoff Merge branch 'develop' of https://
   github.com/jrkolsby/Grape into develop
d8c25d5 2018-12-15 Nick Krasnoff Semant case for empty lists
ab1cdc1 2018-12-15 James Kolsby fix: tests
4605280 2018-12-15 James Kolsby fix: get_outgoing2
9e635a2 2018-12-15 Nick Krasnoff Fixed tests
afa30d7 2018-12-15 James Kolsby fix: reference pointers
30000aa 2018-12-15 James Kolsby fix: nested lists
a85db8a 2018-12-15 James Kolsby fix: data_value
d9774b3 2018-12-15 James Kolsby fix: multi-prop works
efb99a3 2018-12-15 James Kolsby fix: list index
640c379 2018-12-15 James Kolsby broken: outgoing
8809f79 2018-12-15 James Kolsby feat: char index. broken: methods
34c3b25 2018-12-15 James Kolsby feat: nested props / indexes
eb911ff 2018-12-14 James Kolsby feat: grape lib, string equals
bfbf216 2018-12-14 James Kolsby feat: properties
05b4be4 2018-12-14 James Kolsby feat: list index overloading
23b3e40 2018-12-13 James Kolsby feat: list index
2298d00 2018-12-13 James Kolsby fix: directed edges
```

```
f40f12a 2018-12-13 James Kolsby feat: new edge syntax
cabf3c0 2018-12-13 James Kolsby fix: ast and sast strings and tests
f1a5d1c 2018-12-13 HyunBin Yoo feat: debugged all graph functions &
   implemented GraphFind function
86d0d3f 2018-12-13 James Kolsby feat: ignore fail files
b2eecb5 2018-12-13 James Kolsby Merge branch 'master' of https://
   github.com/jrkolsby/Grape
1791323 2018-12-13 James Kolsby Merge pull request #1 from jrkolsby/
   declarations
0733b5c 2018-12-13 James Kolsby fix: tests (origin/declarations,
   declarations)
d5fcfc4 2018-12-12 James Kolsby fix: tests
74e3820 2018-12-12 James Kolsby fix: parse conflicts
1d1ddd3 2018-12-12 James Kolsby fix: dfa.grp
e201e1f 2018-12-12 Nick Krasnoff Working DFA test
ef244a4 2018-12-12 James Kolsby temp: needs empty declaration
d8e7457 2018-12-12 James Kolsby feat: declarations
1e03dbd 2018-12-12 James Kolsby temp: added declaration
57fe1f4 2018-12-12 James Kolsby fix: dfa.grp
56f1923 2018-12-12 James Kolsby temp: needs empty declaration
b08da96 2018-12-12 HyunBin Yoo Merge branch 'master' of https://github.com/
   jrkolsby/Grape
b380e33 2018-12-12 HyunBin Yoo feat: commented out remove
5ff9174 2018-12-12 Nick Krasnoff Working DFA test
9a9d20a 2018-12-12 James Kolsby feat: declarations
0e569a8 2018-12-12 James Kolsby temp: added declaration
dae5546 2018-12-12 James Kolsby fix: dfa.grp
b6712c1 2018-12-12 Nick Krasnoff Dfa testing
92b2389 2018-12-12 Tim Merge branch 'master' of https://github.com/
   jrkolsby/Grape
9ae49c1 2018-12-12 Tim str_equal
7e9418c 2018-12-12 Nick Krasnoff Refixed get_outgoing
6eea7e5 2018-12-12 HyunBin Yoo Merge branch 'master' of https://github.com/
   jrkolsby/Grape
0a27ad1 2018-12-12 HyunBin Yoo feat: added str_equal
db107d2 2018-12-12 Nick Krasnoff Added test files
01e677c 2018-12-12 Tim get_char sig
b95b23f 2018-12-12 Tim scall errors
d297d47 2018-12-12 Tim Merge branch 'master' of https://github.com/
   jrkolsby/Grape
09bd4e7 2018-12-12 Tim str_size error
829b13a 2018-12-12 Nick Krasnoff Modified dfa test
```

```
3ad04ad 2018-12-12 Tim temp 2 dfa
0efe531 2018-12-12 Tim temp comit for dfa
628f294 2018-12-12 HyunBin Yoo Merge branch 'master' of https://github.com/
   jrkolsby/Grape
94c6b28 2018-12-12 HyunBin Yoo feat: working on graph functions
7c09312 2018-12-12 HyunBin Yoo feat: string length
85f0de7 2018-12-12 Tim dfa test session
44eaf4f 2018-12-12 Nick Krasnoff Modified dfa.grp test
c8fd58e 2018-12-12 Nick Krasnoff Merge branch 'master' of https://
   github.com/jrkolsby/Grape
0aee78c 2018-12-12 Nick Krasnoff updated test; get_outgoing finds when a
   node points to itself
8e3a983 2018-12-12 Tim Merge branch 'master' of https://github.com/
   jrkolsby/Grape
8da79bc 2018-12-12 Tim semant declarations
f362902 2018-12-12 HyunBin Yoo deleted
f798af7 2018-12-12 HyunBin Yoo Merge branch 'master' of https://github.com/
   jrkolsby/Grape
0c93bdd 2018-12-12 HyunBin Yoo feat: deleted print
5d1e36f 2018-12-12 Nick Merge branch 'master' of https://github.com/
   jrkolsby/Grape
2b29209 2018-12-12 Nick Add edges to node's list of edges
dde7c32 2018-12-12 HyunBin Yoo added the header file
6b5daf9 2018-12-12 HyunBin Yoo renamed string file
ccc5e37 2018-12-12 HyunBin Yoo feat: string function
fa4770f 2018-12-12 Nick Fixed typo in graph.c
cf11320 2018-12-12 Nick Merge branch 'master' of https://github.com/
   jrkolsby/Grape
eaf7c7c 2018-12-12 Nick C functions to access Nodes and Edges
1912017 2018-12-12 James Kolsby fix: dfa test case
37ff0aa 2018-12-12 James Kolsby feat: dfa test
d6cc30f 2018-12-12 Nick C for getting nodes and edges in .h
74112ff 2018-12-12 Nick C for getting nodes and edges
6d06ab4 2018-12-12 Tim add get str and get int funciton to list
0326707 2018-12-11 Nick Krasnoff Added C code to get element from a list
c355387 2018-12-10 James Kolsby fix: graph init
18b45a6 2018-12-09 Nick Krasnoff Fixed List creation and Makefiles now
   compile with debug info
50bc7b2 2018-12-09 Tim not fixed: list type, infinite loop when ./out.g
968b79d 2018-12-09 Tim change push_list return and type list in codegen
bd27ba7 2018-12-09 Nick Krasnoff Changed list.grp test case
503c5be 2018-12-09 Tim fixed typing of list in codegen
```

```
2a051b0 2018-12-06 Tim feat: list type in codegen
e4aea9d 2018-12-06 Tim merge conflict
7929b30 2018-12-06 Tim codegen.ml
b2af8ec 2018-12-06 HyunBin Yoo missed types.h
1c8877d 2018-12-06 HyunBin Yoo Merge branch 'master' of https://github.com/
   jrkolsby/Grape
9487ed3 2018-12-06 HyunBin Yoo feat adding node and edge to graph
b1115e0 2018-12-06 James Kolsby feat: add edge codegen
f9c2928 2018-12-06 HyunBin Yoo forgot to push list.h
4a9b8d5 2018-12-06 HyunBin Yoo Merge branch 'master' of https://github.com/
   jrkolsby/Grape
09e1979 2018-12-06 HyunBin Yoo fixed errors
cac5f27 2018-12-06 James Kolsby feat: init_graph
4a35830 2018-12-06 HyunBin Yoo feat: list implementation
fd9b22e 2018-12-06 HyunBin Yoo Merge branch 'master' of https://github.com/
   jrkolsby/Grape
9ddbb8b 2018-12-06 HyunBin Yoo feat: basic graph
e80b9d7 2018-12-06 James Kolsby feat: node, edge, graph function calls
1cd9dcc 2018-12-06 James Kolsby feat: ID in graph
dd9389b 2018-12-06 Tim fixed: edge type
d00ba05 2018-12-06 HyunBin Yoo fixed error in graph.c
614f7f1 2018-12-04 HyunBin Yoo feat: function prototypes for graph node
   value and edge
df1e086 2018-12-04 HyunBin Yoo feat: GraphInitNode
aa85177 2018-12-04 James Kolsby fix: migrated tests to dist
0e988f9 2018-12-04 James Kolsby fixed gitignore again
eabd7b2 2018-12-04 James Kolsby fix: run context
1eddd80 2018-12-04 James Kolsby fixed untracked files
a68b5e4 2018-12-04 James Kolsby feat: refactored stdlib
6a3253e 2018-12-03 James Kolsby fix: makefile
045763e 2018-12-03 James Kolsby feat: dist refactor
5073fde 2018-12-03 James fix: link tests to c library
69c3c78 2018-12-03 James feat: linking
b7ff82a 2018-12-03 James feat: node_init
90a5c2b 2018-12-03 James Kolsby feat: refactor README
8db9bd1 2018-12-03 James Kolsby fix: typos and merlin
a269ba2 2018-12-03 HyunBin Yoo feat: implemented leaves, adjacent in C
c40793a 2018-12-03 HyunBin Yoo feat implemented graph functions in C
16158ac 2018-12-01 Nick Krasnoff Semantically checked Graphs
1e29650 2018-12-01 Nick Krasnoff Semant for Graph, Assign tries int =
   Node<Int>
```

```
48eb59e 2018-12-01 Tim Fix(undone): parser Graphlit order fixed, Graph in
   semant bad
0b2d1ba 2018-11-30 Tim fixed: edge case for graph in semant
ece49b2 2018-11-30 Tim feat: check edge case in semant
a04e707 2018-11-30 Tim Merge branch 'master' of https://github.com/
   jrkolsby/Grape
16a08b2 2018-11-30 Nick Krasnoff Compile grape c file
a3e4427 2018-11-30 Tim Merge branch 'master' of https://github.com/
   jrkolsby/Grape
4d5a12c 2018-11-30 Nick Krasnoff Changed semant built in decls format
a818e96 2018-11-30 Nick Krasnoff Added Graph in Semant: now compiles
39f6156 2018-11-30 Tim Merge branch 'master' of https://github.com/
   jrkolsby/Grape
435bec7 2018-11-30 Tim feat: add to codegen and grape.c
7dc1a7d 2018-11-30 Nick Krasnoff Reremoved dict
e5fbd68 2018-11-30 James Kolsby temp
311457d 2018-11-30 Nick Krasnoff Removed colon from scanner
645c41b 2018-11-30 Nick Krasnoff Merge branch 'master' of https://
   github.com/jrkolsby/Grape
c4f8050 2018-11-30 Nick Krasnoff Removed Dicts
e04ce99 2018-11-29 Tim added SList and SNode type
f175850 2018-11-29 Tim fix: reformatted c library, and add to Readme
bde9061 2018-11-29 HyunBin Yoo revised README
c9490fc 2018-11-29 HyunBin Yoo feat: added graph function for root
c827b06 2018-11-29 HyunBin Yoo feat: added function prototypes for graph
   funcitons
4be9e31 2018-11-29 HyunBin Yoo feat: compilable grape.c
4ffa89a 2018-11-29 HyunBin Yoo feat: added structs for graph implementation
2716146 2018-11-29 HyunBin Yoo feat: copy funtion in C
4195544 2018-11-29 HyunBin Yoo fixed a typo in grape.c
31d3b5c 2018-11-29 HyunBin Yoo fixed READMD
a484c66 2018-11-28 HyunBin Yoo feat: list copy in C
2d3da7d 2018-11-28 HyunBin Yoo feat: list copy in C
e137a9c 2018-11-28 Tim Merge branch 'master' of https://github.com/
   jrkolsby/Grape
69a34e7 2018-11-28 Tim feat: list functions in c
6616339 2018-11-28 HyunBin Yoo removed dictionary
56094f7 2018-11-28 HyunBin Yoo combined two c files
0df6cb8 2018-11-28 HyunBin Yoo git removed
b5de2fe 2018-11-26 Nick Krasnoff Graph types pretty print
245e61d 2018-11-26 Nick Krasnoff Graph types in ast/parser
1a272b2 2018-11-26 HyunBin Yoo added comment in list library
```

```
62d4868 2018-11-26 HyunBin Yoo renamed two C library files
84f5fbd 2018-11-26 HyunBin Yoo found this implementation of dictionary in C
7fecc4b 2018-11-26 HyunBin Yoo added reverse function
bde50e9 2018-11-26 HyunBin Yoo C library for list
02fde42 2018-11-26 HyunBin Yoo graphlit
2ae22a9 2018-11-22 James feat: function tests
ffc65f2 2018-11-20 James fix: testing suite
27d8924 2018-11-18 Tim fixed: parser.mly semant.ml in dictlit
2a9dd71 2018-11-18 Tim fixed some ast, sast stuff
bc0d880 2018-11-18 Tim fixed: parser and ast/sast
34ead86 2018-11-15 James feat: broken test script
2e334c6 2018-11-15 James feat: GRAPE V0
99902ed 2018-11-15 James feat: parser
6a7fee6 2018-11-15 James fixes
63244fb 2018-11-14 HyunBin Yoo fixed: Failure
b984e5c 2018-11-14 Tim Merge branch 'master' of https://github.com/
   jrkolsby/Grape
25108b3 2018-11-14 Tim fixed: semant error
e858aa9 2018-11-14 HyunBin Yoo Merge branch 'master' of https://github.com/
   jrkolsby/Grape
5111110 2018-11-14 HyunBin Yoo fixed: raise error for unimplemented
72aa61d 2018-11-14 Nick Krasnoff feat: Declare and Assign only as Statement
3b1246b 2018-11-14 James Kolsby fix: Node types
a0ea84d 2018-11-14 Tim fixed: script
07bdd05 2018-11-14 James Kolsby fix: scripts
35ea92a 2018-11-14 James Kolsby fix: run dev and init
36f8554 2018-11-14 Nick Krasnoff Merge branch 'master' of https://
   github.com/jrkolsby/PLTProject
501c75d 2018-11-14 Nick Krasnoff scanner.mll: Removed IN
01d2d8f 2018-11-14 Nick Krasnoff parser.mly: Added DQUOT
7e1d748 2018-11-14 James Kolsby fix: codegen coverage (james-devops)
0847011 2018-11-14 James Kolsby infra: ./run command
db0730f 2018-11-14 HyunBin Yoo removed unnecessary files
b861327 2018-11-13 Tim feat: dict and possibly graph type checking
42daf57 2018-11-13 Tim fixed: semant merge conflict
0f080a5 2018-11-13 HyunBin Yoo feat: implemented each
ea5193b 2018-11-14 James Kolsby temp
6677fd4 2018-11-14 James Kolsby feat: update makefile
4da6e00 2018-11-13 Tim feat: type checking listlit
6c46b2a 2018-11-13 HyunBin Yoo fixed: Edgelit in semant.ml
c269f2f 2018-11-13 Tim fixed: list type func
e049c20 2018-11-13 Tim working: semant list type checking
```

```
ecda991 2018-11-13 James Kolsby feat: list typechecking
ca0e6c2 2018-11-13 Nick Krasnoff Merge branch 'master' of https://
   github.com/jrkolsby/PLTProject
25d48bb 2018-11-13 Nick Krasnoff prints: String printing in codegen.ml
1a6f4b1 2018-11-13 Tim fixed ast.ml has no typed graph
b42ed0a 2018-11-13 Tim fixed: typing
dc11deb 2018-11-13 Tim fixed: semant changed
87602e1 2018-11-14 James fix: Node semantic checking
ff7ea32 2018-11-13 Nick Krasnoff Merge branch 'master' of https://
   github.com/jrkolsby/PLTProject
8a2eb98 2018-11-13 Nick Krasnoff Updated docker env scripts
0832492 2018-11-13 James Kolsby refactor: new tests
0058ad4 2018-11-13 Tim feat: add typ
7e312c9 2018-11-13 James Kolsby feat: added string_of_typ
2ae87b8 2018-11-13 James Kolsby fix: pretty printing
1a250fd 2018-11-13 Tim fixed: rid of Templates
bfb6d72 2018-11-13 Tim fix: add stuff to pretty print in sast
ea84a41 2018-11-13 Tim feat: update all existing data types in ast
f366115 2018-11-12 Nick Krasnoff Added String type to codegen
64015b9 2018-11-06 Tim fixed: typo in ast.ml
9fac844 2018-11-06 Tim fixed: typo in mehir scripts, clear all conflicts
b0e900b 2018-11-06 Tim fixed: typo in mehir scripts, clear all conflicts
50e0b2d 2018-11-06 James Kolsby refactor: SFloatLit and SIntLit
22759b8 2018-11-05 Tim Merge branch 'master' of https://github.com/
   jrkolsby/Grape
6f5bbfa 2018-11-05 Tim fixed: edge
a740064 2018-11-05 James Kolsby feat: README changes
58bceb3 2018-11-05 HyunBin Yoo completed function
ad8f1a4 2018-11-05 HyunBin Yoo changed function declaration
398dcf2 2018-11-05 Tim Merge branch 'master' of https://github.com/
   jrkolsby/Grape
9348d40 2018-11-05 Tim fixed: conflicts with dean
016c3d0 2018-11-05 James Kolsby refactor: removed .sh from scripts
ced01ee 2018-11-05 HyunBin Yoo Merge branch 'master' of https://github.com/
   jrkolsby/Grape
113aba2 2018-11-05 HyunBin Yoo changed
68815db 2018-11-05 James Kolsby refactor: moved menhir script
22e3200 2018-11-05 James Kolsby feat: decimal syntax
f160627 2018-11-05 James Kolsby feat: floats
56697d4 2018-11-05 Tim Merge branch 'master' of https://github.com/
   jrkolsby/Grape
8718b98 2018-11-05 Tim fixed: expr_opt in stmt
```

```
c86f8b5 2018-11-05 James Kolsby feat: make menhir
60fc38e 2018-11-05 Tim fixed expr_opt in parser.mly
51d823b 2018-11-05 Nick Krasnoff Merge branch 'master' of https://
   github.com/jrkolsby/PLTProject
5e4fcdb 2018-11-05 Nick Krasnoff merge
fcb85a8 2018-11-05 James Kolsby fix: make parser
de6afbd 2018-11-05 Nick Krasnoff Dict in parser/ast
153318c 2018-11-05 HyunBin Yoo changed
2018235 2018-11-05 James Kolsby feat: SAST.ml
d5efc67 2018-11-05 Tim fixed: typo in ast.ml
dd4c541 2018-11-05 James Kolsby feat: documentation
4de7d49 2018-11-05 James Kolsby fix: Makefile compilation (broken)
c7465cf 2018-11-05 Tim feat: added giraph whole folder for reference
b0b3d9c 2018-11-05 Tim feat: add Exp, Float, Dist to ast, and fixed typo
05f9880 2018-11-05 Tim feat: added AMP syntax
43e2f15 2018-11-04 Tim added: menhir shellscript
cbcb18f 2018-11-04 Tim fix: edge declaration is now with underscore
c03b89e 2018-11-04 Tim fix typo in parser.mly and update README.md
a0d8407 2018-11-04 Tim feat: add giraph to git
055279e 2018-11-02 Tim feat: add script to attach container
2cd69ea 2018-11-02 James Kolsby fix: Makefile
85be8a3 2018-11-02 Tim thinking: edge template not working
c46f982 2018-11-02 James Kolsby feat: README changes
ab57a5f 2018-11-01 James Kolsby feat: codegen, semant, grape
1dc100f 2018-11-01 James Kolsby refactor: scripts folder
f285808 2018-11-01 James Kolsby refactor: docs folder
3c49921 2018-11-01 James Kolsby feat: Added dev environment
ea92f7f 2018-11-01 Tim feat: list expr
7580199 2018-11-01 Tim fixed: conflict
dce1b9e 2018-11-01 Tim feat: allExpr to encompass node/edgeexpr,
   implemented template (not working for edges)
baee0fc 2018-11-01 HyunBin Yoo Merge branch 'master' of https://github.com/
   jrkolsby/Grape
4958169 2018-11-01 HyunBin Yoo fixed
979577e 2018-11-01 Tim Merge branch 'master' of https://github.com/
   jrkolsby/Grape
cef616a 2018-11-01 Tim feat: vdecl with value in a single line
265d950 2018-11-01 HyunBin Yoo fixed
b2eb4a5 2018-11-01 Tim feat: added additional operators
7806b35 2018-10-29 Nick Changes with Dean
c7e6424 2018-10-29 Tim fix: format in README.md
```

```
3796648 2018-10-25 Tim feat: add square bracket, ampersand to scanner,
   parser but no syntax
3c49a76 2018-10-25 Nick Krasnoff fixed typo in parser.mly
cb25e27 2018-10-25 James Kolsby feat: 'of' types
82d8006 2018-10-22 Timmypoyu Feat: update Readme.md
b92add6 2018-10-15 James Kolsby feat: assignment bindings
8b6e2b9 2018-10-15 James Kolsby feat: finished graph initialization
1534bfd 2018-10-15 Tim fix: merge conflicts
96f7355 2018-10-15 Tim add grapg_list
c12c296 2018-10-15 James Kolsby feat: graph syntax
affd51a 2018-10-15 James Kolsby feat: comment scanning
3e857c3 2018-10-15 James Kolsby feat: graph syntax
c4035bd 2018-10-15 James Kolsby feat: scanner updates
5232991 2018-10-15 James Kolsby feat: Node and Edge types
4ae9ef9 2018-10-15 Tim changed comment to three backslash
73dc0e0 2018-10-14 James Kolsby refactor: moved reference
0c8641e 2018-10-14 James Kolsby feat: Graph templating in LRM
9c960f6 2018-10-13 James Kolsby refactor: moved MicroC to root
d3a24a2 2018-10-13 James Kolsby feat: Started LRM
1476cbd 2018-10-13 James Kolsby feat: string literals
1ec95df 2018-10-13 James Kolsby feat: primitive types
14417c3 2018-10-13 James Kolsby feat: microc parser and scanner
9c29232 2018-10-13 Nick Krasnoff Added calculator Makefile
20fc371 2018-10-13 James Kolsby fix merge
f990a9c 2018-10-13 James Kolsby fix: deleted files
21a5cd0 2018-10-13 Nick Krasnoff Merge branch 'master' of https://
   github.com/jrkolsby/PLTProject
d9cee65 2018-10-13 Nick Krasnoff Added microc source
80d5dac 2018-10-13 James Kolsby fix: refactor
3f3e27e 2018-10-13 James Kolsby feat: proposal
d902c90 2018-09-07 x first commit
```

15. Code

15.1.Scanner (Lex)

```
(* Ocamllex scanner for Grape *)

{ open Parser }

(*Hello*)
let digit = ['0'-'9']
```

```
let decimal = (digit+ '.' digit*)
let letter = ['A'-'Z' 'a'-'z']
let lowerLetter = ['a'-'z']
(* TODO: Floats *)

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "///"    { mComment lexbuf }           (* Comments *)
| "//"     { sComment lexbuf }
| '('      { LPAREN }
| ')'      { RPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
| '['      { LBRACK }
| ']'      { RBRACK }
| ';'      { SEMI }
| "'"      { SQUOT }
| '"'      { DQUOT }
| ','      { COMMA }
| '.'      { DOT }
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| "**"     { EXP }
| '/'      { DIVIDE }
| '%'      { MOD }
| '='      { ASSIGN }
| "=="     { EQ }
| "!="     { NEQ }
| '&'      { AMP }
| '<'      { LT }
| "<<"     { GRAPS }
| ">>"     { GRAPE }
| "<="     { LEQ }
| ">"      { GT }
| "_"      { UNDS }
| ">="     { GEQ }
| "fun"    { FUN }
| "and"    { AND }
| "or"     { OR }
| "not"    { NOT }
| "if"     { IF }
| "else"   { ELSE }
| "each"   { EACH }
| "for"    { FOR }
| "while"  { WHILE }
| "return" { RETURN }
| "Int"    { INT }
| "Edge"   { EDGE }
```

```
| "Node"   { NODE }
| "Dict"   { DICT }
| "Graph"  { GRAPH }
| "List"   { LIST }
| "String" { STR }
| "Bool"   { BOOL }
| "Yes"   { TRUE }
| "No"  { FALSE }
| decimal as lxm            { FLOAT_LIT(lxm) }
| digit+ as lxm             { INT_LIT(int_of_string lxm) }
| '\"' ([^'\"']* as lxm) '\"' { STR_LIT(lxm) }
| lowerLetter (letter | digit | '_')* as lxm { ID(lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

and mComment = parse
| "///" { token lexbuf }
| _ { mComment lexbuf }

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

15.2.Parser (Yacc)

```
/* Ocamlyacc parser for MicroC */

%{ open Ast %}

%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA LBRACK RBRACK GRAPS GRAPE
SQUOT DQUOT DOT UNDS
%token PLUS MINUS TIMES EXP DIVIDE ASSIGN NOT MOD AMP
%token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR
%token RETURN IF ELSE EACH WHILE FOR FUN THIS
%token INT NODE EDGE GRAPH STR BOOL LIST DICT
%token <int> INT_LIT
%token <string> FLOAT_LIT
%token <string> STR_LIT
%token <string> ID
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left OR
%left AND
%left DOT
%left EQ NEQ
```

```
%left LT GT LEQ GEQ
%left AMP
%left PLUS MINUS
%left TIMES DIVIDE MOD
%left EXP
%right NOT NEG

%start program
%type <Ast.program> program

%%

program:
  decls EOF { $1 }

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

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

/* DO ALL Variable Declarations have to come before all STATEMENTS? */

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

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

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

typ:
    INT                         { Int }
  | STR                         { Str }
  | BOOL                        { Bool }
  | LIST LT typ GT              { List($3) }
  | NODE LT typ GT              { Node($3) }
  | EDGE LT typ GT              { Edge($3, Any) }
  | GRAPH LT typ COMMA typ GT   { Graph(Node($3), Edge($5,$3)) }
```

```
stmt_list:
    stmt           { [$1] }
  | 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) }
  | EACH LPAREN expr RPAREN stmt            { Each($3, $5) }
  | WHILE LPAREN expr RPAREN stmt           { While($3, $5) }
  | typ ID ASSIGN expr SEMI                 { Declare($1, $2, Assign($2,
$4))}
  | typ ID SEMI                             { Declare($1, $2, Noexpr) }

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

edge_expr:
  | MINUS MINUS                           { EdgeLit(Noexpr) }
  | MINUS MINUS GT                        { DirEdgeLit(Noexpr) }
  | MINUS literal MINUS GT                { DirEdgeLit($2) }
  | MINUS literal MINUS                   { EdgeLit($2) }

node_expr:
  | SQUOT literal SQUOT                { NodeLit($2) }

literal:
    ID                          { Id($1) }
  | FALSE                       { BoolLit(false) }
  | TRUE                        { BoolLit(true) }
  | FLOAT_LIT                   { FloatLit($1) }
  | STR_LIT                     { StrLit($1) }
  | INT_LIT                     { IntLit($1) }
  | LPAREN expr RPAREN          { $2 }
  | ID LPAREN actuals_opt RPAREN  { Call($1, $3) }
  | literal DOT ID              { Prop($1, $3) }
  | literal LBRACK expr RBRACK    { Index($1, $3) }
  | literal DOT ID LPAREN actuals_opt RPAREN { Method($1, $3, $5) }

expr:
    literal               { $1 }
  | node_expr             { $1 }
  | GRAPS edge_expr GRAPE   { $2 }
  | GRAPS graph_opt GRAPE   { GraphLit($2) }
  | expr PLUS   expr        { Binop($1, Add,   $3) }
  | expr MINUS  expr        { Binop($1, Sub,   $3) }
```

```
  | expr TIMES  expr        { Binop($1, Mult,  $3) }
  | expr DIVIDE expr        { Binop($1, Div,   $3) }
  | expr EQ     expr        { Binop($1, Equal, $3) }
  | expr NEQ    expr        { Binop($1, Neq,   $3) }
  | expr LT     expr        { Binop($1, Less,  $3) }
  | expr LEQ    expr        { Binop($1, Leq,   $3) }
  | expr GT     expr        { Binop($1, Greater, $3) }
  | expr GEQ    expr        { Binop($1, Geq,   $3) }
  | expr AND    expr        { Binop($1, And,   $3) }
  | expr OR     expr        { Binop($1, Or,    $3) }
  | expr EXP    expr        { Binop($1, Exp,   $3) }
  | expr MOD    expr        { Binop($1, Mod,   $3) }
  | expr AMP    expr        { Binop($1, Amp,   $3) }
  | MINUS expr %prec NEG    { Unop(Neg, $2) }
  | NOT expr                { Unop(Not, $2) }
  | ID ASSIGN expr          { Assign($1, $3) }
  | LBRACK actuals_opt RBRACK     { ListLit($2) }

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

/* comma-separated list */
actuals_list:
    expr { [$1] }
  | actuals_list COMMA expr { $3 :: $1 }

/* graph list */
graph_opt:
  /* nothing */ { [] }
  | graph_list { List.rev $1 }

// The following block means that graph_list can be all nodes
// as path_list can be only a single node
graph_list:
    path_list { [List.append (List.rev (List.tl (List.rev $1))) ([List.hd
(List.rev $1)])] }
  | graph_list COMMA path_list { (List.append (List.rev (List.tl (List.rev
$3))) ([List.hd (List.rev $3)])) :: $1 }

graph_vertex:
    node_expr { $1 }
  | ID        { Id($1) }
// | ID ASSIGN node_expr { Declare } TODO: Assign inside graphs

path_list:
    graph_vertex { [($1, Noexpr)] }
  | graph_vertex edge_expr path_list {($1, $2) :: $3}
```

15.3. Abstract Syntax Tree

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

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

type uop = Neg | Not

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

(* variable type declaration *)
type bind = typ * string

type expr =
    IntLit of int
  | FloatLit of string
  | BoolLit of bool
  | NodeLit of expr
  | EdgeLit of expr
  | DirEdgeLit of expr
  | GraphLit of ((expr * expr) list) list
  | ListLit of expr list
  | StrLit of string
  | Id of string
  | Prop of expr * string
  | Method of expr * string * (expr list)
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Assign of string * expr
  | Call of string * expr list
  | Index of expr * expr
  | Noexpr

type stmt =
    Block of stmt list
  | Expr of expr
  | Return of expr
  | Declare of typ * string * expr
  | If of expr * stmt * stmt
  | Each of expr * stmt
  | While of expr * stmt
  (* | DecAsn of typ * string * 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_op = function
    Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
  | Equal -> "=="
  | Neq -> "!="
  | Mod -> "%"
  | Exp -> "**"
  | Amp -> "&"
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "and"
  | Or -> "or"

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

let rec string_of_expr = function
    IntLit(l) -> string_of_int l
  | FloatLit(l) -> l
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | Id(s) -> s
  | Prop(e, p) -> string_of_expr e ^ "." ^ p
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
  | Unop(o, e) -> string_of_uop o ^ " " ^ string_of_expr e
  | Assign(v, e) -> v ^ " = " ^ string_of_expr e
  | NodeLit(e) -> "'" ^ string_of_expr e ^ "'"
  | EdgeLit(e) -> "-" ^ string_of_expr e ^ "-"
  | DirEdgeLit(e) -> "-" ^ string_of_expr e ^ "->"
  | GraphLit(e) -> "<<" ^ String.concat ", " (List.map (function lst ->
    String.concat " " (List.map (function (k, v) -> string_of_expr k ^ " " ^
    string_of_expr v) lst ))e) ^ ">>"
```

```
  | ListLit(e) -> "[" ^ String.concat ", " (List.map string_of_expr
    (List.rev e)) ^ "]"
  | Index(e, i) -> string_of_expr e ^ "[" ^ string_of_expr i ^ "]"
  | StrLit(e) -> "\"" ^ e ^ "\""
  | Call(f, args) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr args) ^ ")"
  | Method(o, f, args) ->
      string_of_expr o ^ f ^ "(" ^ String.concat ", " (List.map
    string_of_expr args) ^ ")"
  | Noexpr -> ""

let rec string_of_typ = function
    Any -> "*"
  | Int -> "Int"
  | Bool -> "Bool"
  | Void -> "Void"
  | Float -> "Float"
  | Str -> "String"
  | Node(t) -> "Node<" ^ string_of_typ t ^ ">"
  | Edge(t, n) -> "Edge<" ^ string_of_typ t ^ "->" ^ string_of_typ n ^ ">"
  | Graph(s,t) -> "Graph<" ^ string_of_typ s ^ ", " ^ string_of_typ t ^ ">"
  | List(t) -> "List<" ^ string_of_typ t ^ ">"

let rec string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
  | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^
    string_of_stmt s
  | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
      string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s
  | Declare(t, id, a) -> string_of_typ t ^ " " ^ (match a with Noexpr -> id
    | _ -> string_of_expr a) ^ ";\n"
  | Each(e, s) -> "each (" ^ string_of_expr e ^ ")" ^ string_of_stmt s

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

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

15.4.Semantically Checked Abstract Syntax Tree (OCaml)

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

open Ast

type sexpr = typ * sx
and sx =
    SIntLit of int
  | SFloatLit of string
  | SBoolLit of bool
  | SId of string
  | SProp of sexpr * string
  | SMethod of sexpr * string * sexpr list
  | SBinop of sexpr * op * sexpr
  | SUnop of uop * sexpr
  | SAssign of string * sexpr
  | SCall of string * sexpr list
  | SNodeLit of sexpr
  | SEdgeLit of sexpr
  | SDirEdgeLit of sexpr
  | SGraphLit of ((sexpr * sexpr) list) list
  | SListLit of sexpr list
  | SStrLit of string
  | SIndex of sexpr * sexpr
  | SNoexpr

type sstmt =
    SBlock of sstmt list
  | SExpr of sexpr
  | SDeclare of typ * string * sexpr
  | SReturn of sexpr
  | SIf of sexpr * sstmt * sstmt
  | SEach of sexpr * sstmt
  | SWhile of sexpr * sstmt

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

type sprogram = bind list * sfunc_decl list

(* Pretty-printing functions *)
```

```
let rec string_of_sexpr (t, e) =
  "(" ^ string_of_typ t ^ " : " ^ (match e with
    SIntLit(l) -> string_of_int l
  | SFloatLit(l) -> l
  | SBoolLit(true) -> "true"
  | SBoolLit(false) -> "false"
  | SNodeLit(e) -> "'" ^ string_of_sexpr e ^ "'"
  | SListLit(e) -> "[" ^ String.concat "," (List.map string_of_sexpr e) ^
   "]"
  | SIndex(e, i) -> string_of_sexpr e ^ "[" ^ string_of_sexpr i ^ "]"
  | SDirEdgeLit(e) -> "-" ^ string_of_sexpr e ^ "->"
  | SEdgeLit(e) -> "-" ^ string_of_sexpr e ^ "-"
  | SGraphLit(e) -> "<" ^ String.concat ", " (List.map (function lst ->
   String.concat " " (List.map (function (k, v) ->  string_of_sexpr k ^ " "
   ^ string_of_sexpr v) lst )) e) ^ ">"
  | SStrLit(e) -> e
  | SId(e) -> e
  | SProp(e, p) -> string_of_sexpr e ^ "." ^ p
  | SBinop(e1, o, e2) ->
      string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_sexpr e2
  | SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
  | SAssign(v, e) -> v ^ " = " ^ string_of_sexpr e
  | SCall(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^ ")"
  | SMethod(o, f, el) ->
      string_of_sexpr o ^ f ^ "(" ^ String.concat ", " (List.map
   string_of_sexpr el) ^ ")"
  | SNoexpr -> ""
                                  ) ^ ")"

let rec string_of_sstmt = function
    SBlock(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_sstmt stmts) ^ "}\n"
  | SExpr(expr) -> string_of_sexpr expr ^ ";\n";
  | SReturn(expr) -> "return " ^ string_of_sexpr expr ^ ";\n";
  | SIf(e, s, SBlock([])) ->
      "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s
  | SIf(e, s1, s2) ->  "if (" ^ string_of_sexpr e ^ ")\n" ^
      string_of_sstmt s1 ^ "else\n" ^ string_of_sstmt s2
  | SWhile(e, s) -> "while (" ^ string_of_sexpr e ^ ") " ^ string_of_sstmt
   s
  | SDeclare(t, id, a) -> string_of_typ t ^ " " ^ (match (snd a) with
   SNoexpr -> id | _ -> string_of_sexpr a) ^ ";\n"
  | SEach(e, s) -> "each (" ^ string_of_sexpr e ^ ")" ^ string_of_sstmt s

let string_of_sfdecl fdecl =
  "fun " ^ string_of_typ fdecl.styp ^ " " ^
  fdecl.sfname ^ "(" ^ String.concat ", " (List.map snd fdecl.sformals) ^
```

```
  ") {\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)
```

15.5.Semantic Checking

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

open Ast
open Sast

module StringMap = Map.Make(String)

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

   Check each global variable, then check each function *)

let check (globals, functions) =

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

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

  check_binds "global" globals;

  (**** Check functions ****)
  let func_key name args = name
  in

  let method_key typ name args = match typ with
      Graph _ -> "graph:" ^ (func_key name args)
    | _ -> ""
```

```
in

(* Collect function declarations for built-in functions: no bodies *)
let built_in_functions =
  let add_bind map (name, args, ret) =
    let key = func_key name args in
    StringMap.add key {
      typ = ret;
      fname = name;
      formals = args;
      body = [] } map
  in List.fold_left add_bind StringMap.empty [
    ("print",     [(Int, "x")],   Any);
    ("size",      [(List (Edge (Str, Int)), "x")], Int);
  ("node_same", [(Node Int, "x"); (Node Int, "y")], Bool);
      ("graph_to_list", [(Graph (Int, Int), "x")], List (Node Int));
      ("neighbor", [(Node Int, "x")], List(Node Int));
      ("distance", [(Node Int, "x"); (Node Int, "y")], Int);
      ("graph_size", [(Graph (Int, Int),  "x")], Int);
      ("list_get", [(Int, "x"); (List(Node Int), "y")], Node Int);
      ("update_at", [(Int, "x"); (List(Node Int), "y"); (Node Int, "z")],
 List (Node Int));
      ("get_val", [(Node Int, "x")], Int);
      ("push_front_list_node", [(Node Int, "x"); (List(Node Int), "y")],
 Void);
       ("update_node", [(Int, "x"); (Node Int, "y")], Node Int)]

in

let built_in_methods =
  let v = Any in
  let add_bind map (name, typ, frm, ret) =
    let key = method_key typ name frm in
    StringMap.add key {
      typ = ret;
      fname = name;
      formals = frm;
      body = [] } map
    in List.fold_left add_bind StringMap.empty [
      ("outgoing", Graph (v, v), [(Node v, "x")], List (Edge (v, v)));
  ]
in

(* Add function name to symbol table *)
let add_func map fd =
  let key = func_key fd.fname fd.formals
  and built_in_err = "function " ^ fd.fname ^ " may not be defined"
  and dup_err = "duplicate function " ^ fd.fname
  and make_err er = raise (Failure er)
```

```
  in match fd with (* No duplicate functions or redefinitions of built-
 ins *)
      _ when StringMap.mem key built_in_functions -> make_err
 built_in_err
    | _ when StringMap.mem key map -> make_err dup_err
    | _ ->  StringMap.add key fd map
in

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

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

let find_method typ name args  =
  let key = method_key typ name args in
  try StringMap.find key built_in_methods
  with Not_found -> raise (Failure ("unrecognized method " ^ key))
in

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

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

  (* Raise an exception if the given rvalue type cannot be assigned to
     the given lvalue type *)
  let rec check_assign lvaluet rvaluet err =
    match (lvaluet, rvaluet) with
        (Edge (a,_), Edge (b,c)) -> Edge (check_assign a b err, c)
      | (Node _, Node a) -> Node a
      | (List a, List Any) -> List a
      | (List a, List b) -> List (check_assign a b err)
      | (Graph (a, b), Graph (c, _)) ->
          Graph (check_assign a c err, b)
      | (a, b) -> if a = b then a else raise (Failure err)
  in

  let rec concat_statements locals = function
      [] -> locals
    | hd::tl -> concat_statements (match hd with
        Declare (t, id, a) -> (t, id) :: locals
```

```
     | _ -> locals) tl
 in

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

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

 let expr_list lst expr =
     let rec helper typ tlist = function
         [] -> (typ, tlist)
       | hd :: _ when (match (typ, fst (expr hd)) with
             (Node a , Node b) -> a <> b
           | (Edge (a, b), Edge (c, d)) -> a <> c && b <> d
           | (a, b) -> a <> b) ->
             raise (Failure ("Type inconsistency with list "))
       | hd :: tl -> helper typ (expr hd :: tlist) tl
     in
   let typ = match (List.length lst) with
       0 -> Any
     | _ -> (fst (expr (List.hd lst))) in
   helper typ [] lst
 in

 let expr_graph graph expr =
   let rec type_of_path ntyp etyp plist = function
       [] -> ((ntyp, etyp), List.rev plist)
     | hd :: _ when fst (expr (fst hd)) <> ntyp ->
       raise (Failure (
         "node type inconsistency with path " ^
         string_of_typ (fst (expr (fst hd))) ^
         string_of_typ ntyp ))
     | hd :: tl when (List.length tl) = 0 && fst (expr (snd hd)) <> Void
 ->
       raise (Failure ("edge type inconsistency with path "))
     | hd :: tl when (List.length tl) <> 0 && fst (expr (snd hd)) <>
 etyp ->
       raise (Failure ("edge type inconsistency with path "))
     | hd :: tl -> type_of_path ntyp etyp (((expr (fst hd)), (expr (snd
 hd)))::plist) tl
   in
```

```
  let rec expr_graph ntyp etyp type_of_path glist = function
      [] -> ((ntyp, etyp), List.rev glist)
    | hd :: _ when fst (type_of_path ntyp etyp [] hd) <> (ntyp, etyp) -
>
      raise (Failure ("path type inconsistency"))
    | hd :: tl -> expr_graph ntyp etyp type_of_path ((snd (type_of_path
ntyp etyp [] hd))::glist) tl
  in

  let edgeType = fst (expr (snd (List.hd (List.hd graph)))) in
  let nodeType = fst (expr (fst (List.hd (List.hd graph)))) in
  expr_graph nodeType edgeType type_of_path [] graph in

(* Return a semantically-checked expression, i.e., with a type *)
let rec expr = function
    IntLit l  -> (Int, SIntLit l)
  | FloatLit l -> (Float, SFloatLit l)
  | BoolLit l  -> (Bool, SBoolLit l)
  | StrLit s   -> (Str, SStrLit s)
  | NodeLit n ->  let (t, d) = expr n in (Node t, SNodeLit (t, d))
  | ListLit l -> let (t, l) = expr_list l expr in (List t, SListLit l)
  | Index (e, i) ->
      let (te, _) as e' = expr e in
      let (ti, _) as i' = expr i in
      if ti != Int then raise (Failure ("list index not integer"))
      else (match te with
          List x -> (x, SIndex (e', i'))
        | Str -> (Str, SIndex (e', i'))
        | _ -> raise (Failure ("not iterable")))
  | GraphLit g -> let ((n, e), l) = expr_graph g expr in
    (Graph (n, e), SGraphLit l)
  | EdgeLit s -> let t = expr s in (Edge ((fst t), Any), SEdgeLit t)
  | DirEdgeLit s -> let t = expr s in (Edge ((fst t), Any), SDirEdgeLit
t)
  | Noexpr     -> (Void, SNoexpr)
  | Id s       -> (type_of_identifier s, SId s)
  | Prop (e, p) ->
    let (et, _) as e' = expr e in
    let pt = (match (et, p) with
        (Node t,      "val") -> t
      | (Edge (t, _), "val") -> t
      | (Edge (_, t), "to") -> Node t
      | (Edge (_, t), "from") -> Node t
      | (_, _) -> raise (Failure ("no such property"))) in
    (pt, SProp (e', p))
  | Assign (var, e) as ex ->
      let lt = type_of_identifier var
      and (rt, e') = expr e in
```

```
      let err = "illegal assignment " ^ string_of_typ lt ^ " = " ^
        string_of_typ rt ^ " in " ^ string_of_expr ex in
      (check_assign lt rt err, SAssign(var, (rt, e')))
  | Unop(op, e) as ex ->
      let (t, e') = expr e in
      let ty = match op with
        Neg when t = Int || t = Float -> t
      | Not when t = Bool -> Bool
      | _ -> raise (Failure ("illegal unary operator " ^
                             string_of_uop op ^ string_of_typ t ^
                             " in " ^ string_of_expr ex))
      in (ty, SUnop(op, (t, e')))
  | Binop(e1, op, e2) as e ->
      let (t1, e1') = expr e1
      and (t2, e2') = expr e2 in
      (* All binary operators require operands of the same type *)
      let same = t1 = t2 in
      (* Determine expression type based on operator and operand types
*)
      let ty = match op with
        Add | Sub | Mult | Div when same && t1 = Int   -> Int
      | Add | Sub | Mult | Div when same && t1 = Float -> Float
      | Equal | Neq            when same               -> Bool
      | Less | Leq | Greater | Geq
                when same && (t1 = Int || t1 = Float) -> Bool
      | And | Or when same && t1 = Bool -> Bool
      | _ -> raise (
          Failure ("illegal binary operator " ^
                   string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
                   string_of_typ t2 ^ " in " ^ string_of_expr e))
      in (ty, SBinop((t1, e1'), op, (t2, e2')))
  | Call(fname, args) ->
      let args' = List.map expr args in
      let fd = find_func fname args' in
      (fd.typ, SCall(fname, args'))
  | Method(obj, mname, args) ->
      let (t, _) as o' = expr obj in
      let args' = List.map expr args in
      let md = find_method t mname args' in
      match (t, mname) with
          (Graph (_, Edge (e, n)), "outgoing") ->
            ((List (Edge (e, n))), SMethod(o', mname, args'))
        | _ -> raise (Failure "no such method")
in

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

```
    in

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

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

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

15.6. Code Generation

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

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

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

Detailed documentation on the OCaml LLVM library:

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

*)

module L = Llvm
module A = Ast
open Sast

module StringMap = Map.Make(String)

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

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

  (* Get types from the LLVM module context *)
  let i32_t      = L.i32_type    context
  and i8_t       = L.i8_type     context
  and str_t      = L.pointer_type (L.i8_type context)
  and i1_t       = L.i1_type     context
  and float_t    = L.double_type context
  and void_t     = L.void_type   context
  and obj_ptr_t  = L.pointer_type (L.i8_type context)
  and void_ptr_t = L.pointer_type (L.i8_type context)
  in

  (* TODO: Make pointers type-dependent? *)
  (* Return the LLVM type for AST node of Grape type *)
  let ltype_of_typ = function
      A.Int   -> i32_t
    | A.Str   -> str_t
    | A.Bool  -> i1_t
    | A.Float -> float_t
    | A.Void  -> void_t
    | A.Any  -> obj_ptr_t
    | A.Node(_)  -> obj_ptr_t
    | A.List(_)  -> obj_ptr_t
    | A.Edge(_,_)  -> obj_ptr_t
    | A.Graph(_,_) -> obj_ptr_t
  in
```

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

(* STD IO *)

(* Function signature / type *)
let printf_t : L.lltype =
    L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
let printf : L.llvalue =
    L.declare_function "printf" printf_t the_module in
let printbig_t : L.lltype =
    L.function_type i32_t [| i32_t |] in
let printbig : L.llvalue =
    L.declare_function "printbig" printbig_t the_module in

(* NODE FUNCTIONS *)

let init_t : L.lltype =
    L.var_arg_function_type obj_ptr_t [| void_ptr_t |] in

let init_node : L.llvalue =
    L.declare_function "init_node" init_t the_module in
let init_edge : L.llvalue =
    L.declare_function "init_edge" init_t the_module in

let init_graph_t : L.lltype =
    L.var_arg_function_type obj_ptr_t [||] in
let init_graph : L.llvalue =
    L.declare_function "init_graph" init_graph_t the_module in

let add_node_t : L.lltype =
    L.var_arg_function_type obj_ptr_t [| obj_ptr_t |] in
let add_node : L.llvalue =
    L.declare_function "add_node" add_node_t the_module in

let add_edge_t : L.lltype =
    L.var_arg_function_type obj_ptr_t [| obj_ptr_t |] in
let add_edge : L.llvalue =
    L.declare_function "add_edge" add_edge_t the_module in

let link_edge_t : L.lltype =
  L.var_arg_function_type void_t [|obj_ptr_t; obj_ptr_t|] in
```

```
let link_edge_to : L.llvalue =
    L.declare_function "link_edge_to" link_edge_t the_module in

let link_edge_from : L.llvalue =
    L.declare_function "link_edge_from" link_edge_t the_module in

let node_same_t : L.lltype =
    L.var_arg_function_type i1_t [|obj_ptr_t; obj_ptr_t|] in
let node_same : L.llvalue =
    L.declare_function "node_same" node_same_t the_module in


let update_node_t : L.lltype =
    L.var_arg_function_type obj_ptr_t [|i32_t; obj_ptr_t|] in
let update_node : L.llvalue =
    L.declare_function "update_node" update_node_t the_module in

(* This must match the C library function name *)

(* list functions*)

let init_list_t : L.lltype =
    L.var_arg_function_type obj_ptr_t [||] in
let init_list : L.llvalue =
    L.declare_function "init_list" init_list_t the_module in

let push_list_t : L.lltype =
    L.var_arg_function_type void_t [|obj_ptr_t; void_ptr_t |] in
let push_list : L.llvalue =
    L.declare_function "push_list" push_list_t the_module in

let push_front_list_t : L.lltype =
    L.var_arg_function_type void_t [|obj_ptr_t; void_ptr_t|] in
let push_front_list : L.llvalue =
    L.declare_function "push_front_list" push_front_list_t the_module in
let push_front_list_node : L.llvalue =
    L.declare_function "push_front_list" push_front_list_t the_module in



let pop_front_list_t : L.lltype =
    L.var_arg_function_type void_t [|obj_ptr_t|] in
let pop_front_list : L.llvalue =
    L.declare_function "pop_front_list" pop_front_list_t the_module in

let pop_list_t : L.lltype =
    L.var_arg_function_type void_t [|obj_ptr_t|] in
let pop_list : L.llvalue =
    L.declare_function "pop_list" pop_list_t the_module in
```

```
let list_get_t : L.lltype =
    L.var_arg_function_type void_ptr_t [|i32_t ; obj_ptr_t|] in
let list_get : L.llvalue =
    L.declare_function "list_get" list_get_t the_module in


let size_t : L.lltype =
    L.var_arg_function_type i32_t [|obj_ptr_t|] in
let size : L.llvalue =
    L.declare_function "size" size_t the_module in
let update_at_t : L.lltype =
    L.var_arg_function_type obj_ptr_t[|i32_t ; obj_ptr_t ; void_ptr_t|]
 in
let update_at : L.llvalue =
    L.declare_function "update_at" update_at_t the_module in

(* string functions*)

let get_char_t: L.lltype =
    L.var_arg_function_type str_t [|i32_t; str_t|] in
let get_char : L.llvalue =
    L.declare_function "get_char" get_char_t the_module in

let str_size_t : L.lltype =
    L.var_arg_function_type i32_t [|str_t|] in
let str_size : L.llvalue =
    L.declare_function "str_size" str_size_t the_module in

let str_equal_t : L.lltype =
    L.var_arg_function_type i1_t [|str_t; str_t|] in
let str_equal : L.llvalue =
    L.declare_function "str_equal" str_equal_t the_module in


(* graph functions*)


let get_outgoing_t : L.lltype =
    L.var_arg_function_type obj_ptr_t [|obj_ptr_t|] in
let get_outgoing : L.llvalue =
    L.declare_function "get_outgoing" get_outgoing_t the_module in

let get_outgoing2_t : L.lltype =
    L.var_arg_function_type obj_ptr_t [|obj_ptr_t; obj_ptr_t|] in
let get_outgoing2 : L.llvalue =
    L.declare_function "get_outgoing2" get_outgoing2_t the_module in

let get_t : L.lltype =
```

```
    L.var_arg_function_type void_ptr_t [|obj_ptr_t|] in
 let get_val : L.llvalue =
    L.declare_function "get_val" get_t the_module in

 let get_to_t : L.lltype =
    L.var_arg_function_type obj_ptr_t [|obj_ptr_t|] in
 let get_to : L.llvalue =
    L.declare_function "get_to" get_to_t the_module in

 let get_from_t : L.lltype =
    L.var_arg_function_type obj_ptr_t [|obj_ptr_t|] in
 let get_from : L.llvalue =
    L.declare_function "get_from" get_from_t the_module in

let graph_to_list_t : L.lltype =
    L.var_arg_function_type obj_ptr_t [|obj_ptr_t|] in
 let graph_to_list : L.llvalue =
    L.declare_function "graph_to_list" graph_to_list_t the_module in

let neighbor_t : L.lltype =
    L.var_arg_function_type obj_ptr_t [|obj_ptr_t|] in
 let neighbor : L.llvalue =
    L.declare_function "neighbor" neighbor_t the_module in

let distance_t : L.lltype =
    L.var_arg_function_type i32_t [|obj_ptr_t; obj_ptr_t|] in
 let distance : L.llvalue =
    L.declare_function "distance" distance_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 str_format_str = L.build_global_stringptr "%s\n" "fmt" builder
 and float_format_str = L.build_global_stringptr "%g\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 add_declaration locals = function
       [] -> locals
     | hd::tl -> add_declaration (match hd with
          SDeclare (t, id, _) -> (t, id) :: locals
        | _ -> locals) tl
   in
       (* slocals: typ * string list *)
   let declarations = add_declaration [] fdecl.sbody in
   List.fold_left add_local formals declarations
 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 ((typ, e) : sexpr) = match e with
   SIntLit i   -> L.const_int i32_t i
 | SStrLit s   -> L.build_global_stringptr s "string" builder
 | SBoolLit b  -> L.const_int i1_t (if b then 1 else 0)
```

```
| SFloatLit l -> L.const_float_of_string float_t l
| SNoexpr     -> L.const_int i32_t 0
| SId s       -> L.build_load (lookup s) s builder
| SProp (o, p) ->
  let tobj = fst o in
  let o' = expr builder o in
  (match (tobj, p) with
    (A.Edge (t, _), "val") ->
      let dest_ptr = L.pointer_type (ltype_of_typ t) in
      let data_ptr = L.build_call get_val [|o'|] "val" builder in
      let data_ptr = L.build_bitcast data_ptr dest_ptr "data" builder
in
      L.build_load data_ptr "data" builder
  | (A.Node t, "val") ->
      let dest_ptr = L.pointer_type (ltype_of_typ t) in
      let data_ptr = L.build_call get_val [|o'|] "val" builder in
      let data_ptr = L.build_bitcast data_ptr dest_ptr "data" builder
in
      L.build_load data_ptr "data" builder
  | (A.Edge (_, t), "to") ->
      L.build_call get_to [|o'|] "to" builder
  | (A.Edge (_, t), "from") ->
      L.build_call get_from [|o'|] "from" builder
      (*let dest_ptr = L.pointer_type (ltype_of_typ t) in
      let data_ptr = L.build_call get_from [|o'|] "from" builder in
      L.build_bitcast data_ptr dest_ptr "data" builder*)
  | (_, _) -> raise (Failure "no such property"))

| SAssign (s, e) -> let e' = expr builder e in
  ignore(L.build_store e' (lookup s) builder); e'

| SNodeLit (t, v) -> (* Cast data type into void pointer to init node
*)
  let data_value = expr builder (t, v) in
  let data = L.build_malloc (ltype_of_typ t) "data_malloc" builder in
    ignore ( L.build_store data_value data builder);
  let data = L.build_bitcast data void_ptr_t "data_bitcast" builder in
  let node = L.build_call init_node [|data|] "init_node" builder in
node
| SEdgeLit (t, v) ->
  let data_value = expr builder (t, v) in
  let data = L.build_malloc (ltype_of_typ t) "data_malloc" builder in
    ignore ( L.build_store data_value data builder);
  let data = L.build_bitcast data void_ptr_t "data_bitcast" builder in
  let edge = L.build_call init_edge [|data|] "init_edge" builder in
edge
  (* TODO: Initialize with empty lists *)
| SDirEdgeLit _ -> raise (Failure "Unimplemented")
| SGraphLit l ->
```

```
  let graph = L.build_call init_graph [||] "init_graph" builder in
  let rec init_path lastEdge isLast = function
    | [] -> graph
    | [hd] when isLast = 1 ->
      let node = expr builder (fst hd) in
      ignore(L.build_call add_node [|graph; node|] "" builder);
      ignore(L.build_call link_edge_to [|lastEdge; node|] "" builder);
      graph
    | hd::tl when isLast = 0 ->
      let edge = expr builder (snd hd) in
      let node = expr builder (fst hd) in
      ignore(L.build_call add_node [|graph; node|] "" builder);
      ignore(L.build_call add_edge [|graph; edge|] "" builder);
      ignore(L.build_call link_edge_from [|edge; node|] "" builder);
      init_path edge 1 tl
    | hd::tl  ->
      let edge = expr builder (snd hd) in
      let node = expr builder (fst hd) in
      ignore(L.build_call add_node [|graph; node|] "" builder);
      ignore(L.build_call add_edge [|graph; edge|] "" builder);
      ignore(L.build_call link_edge_from [|edge; node|] "" builder);
      ignore(L.build_call link_edge_to [|lastEdge; node|] "" builder);
      init_path edge 1 tl
  in
  ignore(List.map (init_path (L.const_int i8_t 0) 0) l); graph
| SIndex (e, i) ->
  let e' = expr builder e in
  let i' = expr builder i in
  (match (fst e) with
      A.Str -> L.build_call get_char [|i'; e'|] "get_char" builder
    | A.List t ->
      let data_ptr = L.build_call list_get [|i'; e'|] "list_get"
builder in
      match t with
          A.List _ | A.Node _ | A.Edge (_,_) | A.Graph _ -> data_ptr
        | _ ->
          let dest_ptr = L.pointer_type (ltype_of_typ t) in
          let data_ptr = L.build_bitcast data_ptr dest_ptr "data"
builder in
          L.build_load data_ptr "data" builder
    | _ -> raise (Failure "Cannot index type"))
| SListLit i ->
  let rec fill_list lst = (function
  [] -> lst
  |sx :: tail ->
  let (atyp,_) = sx in
  let data_ptr = (match atyp with
    A.List _ | A.Graph (_,_) | A.Edge _ | A.Node _  -> expr builder sx
```

```
    | _  -> let data = L.build_malloc (ltype_of_typ atyp) "data"
builder in
      let data_value = expr builder sx in
      ignore (L.build_store data_value data builder); data) in
  let data = L.build_bitcast data_ptr void_ptr_t "data" builder in
  ignore(L.build_call push_list [|lst; data|] "" builder); fill_list
lst tail) in
  let lst = L.build_call init_list [||] "init_list" builder in
  fill_list lst i
| SBinop ((A.Float,_ ) as e1, op, e2) ->
  let e1' = expr builder e1
  and e2' = expr builder e2 in
  (match op with
      A.Add     -> L.build_fadd
    | A.Sub     -> L.build_fsub
    | A.Mult    -> L.build_fmul
    | A.Div     -> L.build_fdiv
    | A.Exp     -> raise (Failure "Unimplemented")
    | A.Mod     -> raise (Failure "Unimplemented")
    | A.Amp     -> raise (Failure "Unimplemented")
    | A.Equal   -> L.build_fcmp L.Fcmp.Oeq
    | A.Neq     -> L.build_fcmp L.Fcmp.One
    | A.Less    -> L.build_fcmp L.Fcmp.Olt
    | A.Leq     -> L.build_fcmp L.Fcmp.Ole
    | A.Greater -> L.build_fcmp L.Fcmp.Ogt
    | A.Geq     -> L.build_fcmp L.Fcmp.Oge
    | A.And | A.Or ->
      raise (Failure "internal error: semant should have rejected and/
or on float")
  ) e1' e2' "tmp" builder
| SBinop ((A.Str, _) as e1, A.Equal, e2) ->
  let e1' = expr builder e1
  and e2' = expr builder e2 in
  L.build_call str_equal [|e1'; e2'|] "str_equal" builder
| SBinop (e1, op, e2) ->
  let e1' = expr builder e1
  and e2' = expr builder e2 in
  (match op with
      A.Add     -> L.build_add
    | A.Sub     -> L.build_sub
    | A.Exp     -> raise(Failure "Unimplemented")
    | A.Mod     -> raise(Failure "Unimplemented")
    | A.Amp     -> raise(Failure "Unimplemented")
    | A.Mult    -> L.build_mul
    | A.Div     -> L.build_sdiv
    | A.And     -> L.build_and
    | A.Or      -> L.build_or
    | A.Equal   -> L.build_icmp L.Icmp.Eq
    | A.Neq     -> L.build_icmp L.Icmp.Ne
```

```
        | A.Less    -> L.build_icmp L.Icmp.Slt
        | A.Leq     -> L.build_icmp L.Icmp.Sle
        | A.Greater -> L.build_icmp L.Icmp.Sgt
        | A.Geq     -> L.build_icmp L.Icmp.Sge
      ) 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
    | SCall ("print", [e]) ->
        let e' = expr builder e in
        (match fst e with
          Int -> L.build_call printf [| int_format_str; e' |] "printf"
   builder
        | Str -> L.build_call printf [| str_format_str; e' |] "prints"
   builder
        | Bool -> L.build_call printf [| int_format_str; e' |] "printb"
   builder
        | Float -> L.build_call printf [| float_format_str; e' |] "printf"
   builder)
    | SCall ("size", [e]) ->
        let e' = expr builder e in
        (match fst e with
          List _ -> L.build_call size [|e'|] "size" builder
        | Str -> L.build_call str_size [|e'|] "str_size" builder)
    | SCall ("node_same", [e;f]) -> L.build_call node_same [|expr builder
   e; expr builder f|] "node_same" builder
    | SCall ("graph_to_list", [e]) -> L.build_call graph_to_list [|expr
   builder e|] "graph_to_list" builder
    | SCall ("neighbor", [e]) -> L.build_call neighbor [|expr builder e|]
   "neighbor" builder
    | SCall ("distance", [e;f]) -> L.build_call distance [|expr builder e;
   expr builder f|] "distance" builder
    | SCall ("push_front_list_node", [e;f]) -> L.build_call push_front_list
   [|expr builder e; expr builder f|] "" builder
 | SCall ("list_get", [e;f]) -> L.build_call list_get [|expr builder e;
   expr builder f|] "list_get" builder
 | SCall ("update_at", [e;f;g]) -> L.build_call update_at [|expr builder e;
   expr builder f; expr builder g|] "" builder
| SCall ("get_val", [e]) -> let data_ptr = L.build_call get_val [|expr
   builder e|] "get_val" builder in
          let data_ptr = L.build_bitcast data_ptr (L.pointer_type i32_t)
   "data" builder in
                L.build_load data_ptr "data" builder
 | SCall ("update_node", [e;f]) -> L.build_call update_node [|expr builder
   e; expr builder f|] "update_node" builder
```

```
    | SCall (f, args) ->
        let (fdef, fdecl) = StringMap.find f function_decls in
        let llargs = List.rev (List.map (expr builder) args) in
        let result = (match fdecl.styp with
            A.Void -> ""
          | _ -> f ^ "_result") in
        L.build_call fdef (Array.of_list llargs) result builder

    | SMethod (o, m, args) ->
      let tobj = fst o in
      let o' = expr builder o in
      (match (tobj, m) with
        (A.Graph (n, e), "outgoing") ->
          let a' = expr builder (List.hd args) in
          let ltype = ltype_of_typ (A.List (A.Edge (e, n))) in
          let data_ptr = L.build_call get_outgoing2 [|o'; a'|] "outgoing"
   builder in
          L.build_bitcast data_ptr ltype "data" builder
        | (_, _) -> raise (Failure "no such method"))
  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
  | SDeclare (_, _, a) -> ignore(expr builder a); builder
  | SReturn e -> ignore(match fdecl.styp with
        A.Void -> L.build_ret_void builder          (* return void instr *)
      | _ -> L.build_ret (expr builder e) builder );  (* Build return
   statement *)
        builder
  | SIf (predicate, then_stmt, else_stmt) ->
    let bool_val = expr builder predicate in
    let merge_bb = L.append_block context "merge" the_function in
    let build_br_merge = L.build_br merge_bb in (* partial function *)
```

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

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

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

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

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

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

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

  | SEach (_, _) -> raise(Failure "not implemented!")
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
```

15.7.C Standard Library

```
// string.c
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include "grapestring.h"

char *get_char (int b, char *a) {

        if (b >= strlen(a)) {
                exit(1);
        }
        static char str[2]= "\0";
        str[0] = a[b];
        return str;
}

int str_size(char *a) {

        return strlen(a);
}

// 0 --> false
// everything else --> true

bool str_equal(char *a, char *b) {
        if (strcmp(a, b) != 0) {
                return false;
        } else {
                return true;
        }
}
// int main() {
        /* char *test = "hello world"; */
        /* printf("%s", get_char(test, 0)); */
        /* printf("%s", get_char(test, 1)); */
        /* printf("%s", get_char(test, 2)); */
        /* printf("%s", get_char(test, 3)); */
        /* printf("%s", get_char(test, 4)); */
        /* printf("%s", get_char(test, 5)); */
        /* printf("%s", get_char(test, 6)); */
        /* printf("%d", str_size(test)); */
        /* char *test2 = "h!"; */
        /* if (str_equal(test, test)) { */
        /*      printf("passed1"); */
        /* } */
        /* if (!str_equal(test, test2)) { */
        /*      printf("passed2"); */
        /* } */
        /* return 0; */
//}
```

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

#include "list.h"

struct Node *init_node(void *input) {
        struct Node *node = (struct Node *)malloc(sizeof(struct Node));
        node->data = input;
        node->edges = init_list();
        return node;
}

struct Graph *init_graph() {
        struct Graph *graph = (struct Graph *)malloc(sizeof(struct Graph));
        graph->nodes = init_list();
        graph->edges = init_list();
        return graph;
}

struct Edge *init_edge(void *data) {
        struct Edge *edge = (struct Edge *)malloc(sizeof(struct Edge));
        edge->data = data;
        edge->to = NULL;
        edge->from = NULL;
        return edge;
}

void link_edge_from(struct Edge *e, struct Node *from) {
        e->from = from;
        push_list(from->edges, e);
}

void link_edge_to(struct Edge *e, struct Node *to) {
        e->to = to;
        push_list(to->edges, e);
}

void add_node(struct Graph *graph, struct Node *node) {
        struct ListNode *ncheck = graph->nodes->head;
        while( ncheck ){
                if( ncheck->data == node) return;
                ncheck = ncheck->next;
        }
        push_list(graph->nodes, node);
}
```

```
void add_edge(struct Graph *graph, struct Edge *edge) {
        push_list(graph->edges, edge);
}

void *get_val(struct Node *node) {
    return node->data;
}

struct Node *get_to(struct Edge *edge) {
    return edge->to;
}

struct Node *get_from(struct Edge *edge) {
    return edge->from;
}

struct List *get_outgoing(struct Node *node) {
    struct List *adj = node->edges;
    struct List *outgo = init_list();
    struct ListNode *lnode = adj->head;
    struct Node *tnode;
    struct Edge *tedge;
    while( lnode ) {
        tedge = (struct Edge *)lnode->data;
        tnode = (struct Node *)tedge->from;
        if( tnode == node )
            push_list(outgo, tedge);
        lnode = lnode->next;
    }
    return outgo;
}

struct List *get_outgoing2(struct Graph *graph, struct Node *node) {
    struct List *adj = graph->edges;
    struct List *outgo = init_list();
    struct ListNode *lnode = adj->head;
    struct Node *tnode;
    struct Edge *tedge;
    while( lnode ) {
        tedge = (struct Edge *)lnode->data;
        tnode = (struct Node *)tedge->from;
        if( tnode == node )
            push_list(outgo, tedge);
        lnode = lnode->next;
    }
    return outgo;
}

int GraphSize(struct Graph *graph) {
```

```
        return size(graph->nodes);
}

bool GraphIsEmpty(struct Graph *graph) {

        return ((graph->nodes)->head == 0);
}

struct List *GraphLeaves(struct Graph *graph) {

        struct List *leavesList = (struct List *)malloc(sizeof(struct
   List));
        struct ListNode *target = (graph->nodes)->head;
        while (target) {
                struct Node *targetGraph = (struct Node *)(target->data);
                struct List *edges = targetGraph->edges;
                struct ListNode *tedge = edges->head;
                while (tedge) {
                        struct Node *anode = ((struct Edge *)(tedge-
   >data))->to;
                        if (anode == NULL) {
                                push_list(leavesList, anode);
                                tedge = tedge->next;
                        }
                target = target->next;

                }
        }
        return leavesList;
}

struct List *GraphAdjacent(struct Graph *graph, struct Node *node) {

        struct List *adjacentList = (struct List *)malloc(sizeof(struct
   List));
        struct ListNode *target = (graph->nodes)->head;
        while (target) {
                if (node->data == ((struct Node *)(target->data))->data) {
                        struct Node *targetGraph = (struct Node *)(target-
   >data);
                        struct List *edges = targetGraph->edges;
                        struct ListNode *tedge = edges->head;
                        while (tedge) {
                                struct Node *anode = ((struct Edge *)
   (tedge->data))->to;
                                push_list(adjacentList, anode);
                                tedge = tedge->next;
                        }
                }
```

```
                target = target->next;

        }
        return adjacentList;
}



// when the node already exists
void GraphAddEdge(struct Graph *graph, void *weight, struct Node *inputTo,
    struct Node *inputFrom, void *value) {

        struct ListNode *node = (graph->nodes)->head;
     while (node) {
                if (node->data == value) { // find the same node and add
    that edge
                        struct Edge *edge = (struct Edge
    *)malloc(sizeof(struct Edge));
                        struct Node *graphNode = node->data;
                        edge->data = weight;
                        edge->to = inputTo;
                        edge->from = inputFrom;
                        push_list(graphNode->edges, edge);
                        break;
                } else {
                        continue;
                }
                node = node->next;
        }
}

bool GraphFind(struct Graph *graph, void *value) {

    struct ListNode *node = (graph->nodes)->head;

    while (node) {
                if (((struct Node *)(node->data))->data == value) {
                        return true;
                }
                node = node->next;
        }
        return false;
}

struct Node *  update_node(int val, struct Node * a){
        free(a->data);
        void *ptr = malloc(sizeof(int));
        *((int*)ptr) = val;
```

```
        a->data = ptr;
        return a;
}


struct List *graph_to_list(struct Graph *graph) {
        return graph->nodes;
}

struct List *neighbor(struct Node *node){
        struct List *list = (struct List *)malloc(sizeof(struct List));
        struct List *edgelist = node->edges;
        struct ListNode *target = edgelist->head;
        while (target) {
                if(((struct Edge *)(target->data))->to != node){
                    push_list(list, (void *)((struct Edge *)(target-
   >data))->to);
                } else {
                    push_list(list, (void *)((struct Edge *)(target-
   >data))->from);
                }
                target = target->next;
        }
        return list;
}
bool node_same(struct Node *a, struct Node *b){
        if (a == b) {
                return true;
        } else {
                return false;
        }

}

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#include "list.h"

struct Node global;

struct List *init_list() {
        struct List *list = (struct List *)malloc(sizeof(struct List));
        list->head = NULL;
        return list;
}

int size(struct List *list) {
```

```
        if (isEmptyList(list)) {
                return 0;
        }
        int i = 1;
        struct ListNode *node = list->head;

        while(node->next) {
                i += 1;
                node = node->next;
        }
        return i;
}

void *list_get(int n, struct List *list){
    if( n >= size(list) ) {
        fprintf(stderr, "List Index out of bounds: given %d for list of
   size %d\n",n,size(list));
        exit(-1);
    }
    struct ListNode *node = list->head;
    if( n == 0 ){
       return node->data;
    }
    for(int i=0; i<n ; i++ ) {
        node = node->next;
    }
    return node->data;
}

int isEmptyList(struct List *list) {
    return (list->head == NULL);
}

void reverseList(struct List *list) {
    struct ListNode *prv = NULL;
    struct ListNode *cur = list->head;
    struct ListNode *nxt;

    while (cur) {
                nxt = cur->next;
                cur->next = prv;
                prv = cur;
                cur = nxt;
    }

    list->head = prv;
}
```

```
void push_list(struct List *list, void *data) {
    struct ListNode *node = (struct ListNode *)malloc(sizeof(struct
   ListNode));

    node->data = data;
    node->next = NULL;

    if (list->head == NULL) {
        list->head = node;
    } else {
        struct ListNode *end = list->head;
        while (end->next != NULL) {
            end = end->next;
        }
        end->next = node;
    }
}

void push_front_list(struct List *list, void *data) {
    struct ListNode *node = (struct ListNode *)malloc(sizeof(struct
   ListNode));

    node->data = data;
    node->next = list->head;
    list->head = node;

}

void pop_front_list(struct List *list) {

        struct ListNode *oldHead = list->head;
    list->head = oldHead->next;
    void *data = oldHead->data;
    free(oldHead);

}

void pop_list(struct List *list) {
        reverseList(list);
        pop_front_list(list);
        reverseList(list);
}


/*
 * I don't think we need to have a traverse function
void traverseList(struct List *list, void (*f)(void *)) {
    struct Node *node = list->head;
    while (node) {
```

```
                f(node->data);
                node = node->next;
    }
}
*/



void removeAllNodes(struct List *list) {
    while (!isEmptyList(list)) {
                pop_front_list(list);
        }
}
/*
struct ListNode *addAfter(struct List *list, struct ListNode *prevNode,
   void *data) {
    if (prevNode == NULL) {
                return addFront(list, data);
        }

    struct ListNode *node = (struct ListNode *)malloc(sizeof(struct
   ListNode));

        if (node == NULL) {
                return NULL;
        }

    node->data = data;
    node->next = prevNode->next;
    prevNode->next = node;

        return node;
}
*/
/*
struct List *copy(struct List *list) {
        struct List *new = (struct List *)malloc(sizeof(struct List));
    struct ListNode *node = list->head;
        struct ListNode *newNode = NULL;
        initList(new);

    while (node) {
                newNode = addAfter(new, newNode, node->data);
                node = node->next;
        }
        reverseList(new);
        return new;

}
```

```
*/

struct List *update_at(int x, struct List *list, void *y) {
        if(isEmptyList(list)) {
                return list;
        }

        struct ListNode *iter_node = list->head;

        ///if index is wrong, return original list
        if( x >= size(list) ) {
                fprintf(stderr, "List Index out of bounds: given %d for
   list of size %d\n",x,size(list));
                exit(-1);
        }

        if(x == 0){
                iter_node->data = y;
                return list;
        }

        for( int i = 0 ; i!=x ; i++ ) iter_node = iter_node->next;

        iter_node->data = y;

        return list;
}


/*
struct List *insert(int x, struct List *list, void *y) {
        if(isEmptyList(list)) {
                return list;
        }

        struct ListNode *node = (struct ListNode *)malloc(sizeof(struct
   ListNode));

        if (node == NULL) {
                return NULL;
        }

        struct ListNode *iter_node = list->head;
        struct ListNode *insert_node = (struct ListNode *) y;

        ///size
        int i = size(list);

        ///if index is wrong, return original list
```

```
        if(size(list) - 1 < x) {
                return list;
        }

        if (x > 0) {
                x -= 1;
                iter_node = iter_node->next;
        }

        insert_node->data = iter_node->next->data;
        insert_node->next = iter_node->next;
        iter_node->next = insert_node;

        return list;
}*/

bool isEqual(struct ListNode *a, struct ListNode *b) {

        if (a == NULL || b == NULL) {
                return NULL;
        }

        int *dataA = (int *) a->data;
        int *dataB = (int *) b->data;

        if (*dataA == *dataB) {
                return 1;
        }

        return 0;
}

struct List *list_remove(struct List *list, void *y) {

        if (isEmptyList(list)) {
                return NULL;
        }

        struct ListNode *node = list->head;

        while (!isEqual(node, y)) {
                node = node->next;
        }

        if (node != NULL) {
                node->next = node->next->next;
        }

        return list;
```

```
}
```

Grape Standard Library

```
fun Bool dfa(Graph<Bool, String> dfa, Node<Bool> start, String input) {

    Int i = 0;
    Int token = 0;

    String character;

    Bool found = No;
    Edge<String> aEdge;

    Node<Bool> state = start;

    List<Edge<String> > transitions;

    while(token < size(input)) {

        character = input[token];
        transitions = dfa.outgoing(state);

        i = 0;
        while ((i < size(transitions)) and (not found)) {
            if (transitions[i].val == character) {
                state = transitions[i].to;
                found = Yes;
            }
            i = i + 1;
        }
        if (not found) {
            return No;
        }

        found = No;
        token = token + 1;
    }

    return state.val;
}
```

Test Files

Grape Executable

```
#!/bin/bash

HOME=$(dirname "$0")
PROGRAM="$1"
OBJECT="${1%.*}.s"
OUTFILE=${2:-"./out.g"}

cat $HOME/grplib.grp "$1" | $HOME/grape.native | llc -relocation-model=pic
    > $OBJECT
cc -g -O -o $OUTFILE $OBJECT $HOME/stdlib.o

rm $OBJECT
chmod 750 $OUTFILE
```

Makefile

```
.PHONY : all
all : grape.native stdlib.o
        cp grape ../dist
        cp stdlib.o ../dist
        cp stdlib/grplib.grp ../dist
        cp _build/grape.native ../dist

stdlib.o:
        cd stdlib && gcc -g -O -c ./*.c
        ld -r stdlib/*.o -o stdlib.o

grape.native :
        ocamlbuild -use-ocamlfind -tag debug -r \
                -pkgs llvm,llvm.analysis \
                -cflags -w,+a-4 \
                grape.native

.PHONY : clean
clean :
        rm -rf _build grape.native *.o **/*.o parser.ml parser.mli
```

Testing Executables

```
#!/bin/bash


CONTAINER=$(docker ps -a | grep columbiasedwards/plt | head -1 | cut -d' '
    -f1)
docker start $CONTAINER
docker attach $CONTAINER
#!/bin/sh
```

```
docker run -it -v `pwd`:/home/grape -w /home/grape columbiasedwards/plt
#!/bin/bash

menhir --interpret --interpret-show-cst ./src/parser.mly
#!/bin/bash

HOME="$1"
TESTS="./tests/*.grp"
GRAPE="./dist/grape"
GRAPENATIVE="./dist/grape.native"

for TEST in $TESTS
do

    RESULT=$(eval "$GRAPE $TEST && ./out.g" 2>&1)
    AST_RESULT=$(eval "$GRAPENATIVE -a $TEST" 2>&1)
    SAST_RESULT=$(eval "$GRAPENATIVE -s $TEST" 2>&1)

    rm $FAILFILE;
    OUTFILE="${TEST%.*}.out"
    FAILFILE="${TEST%.*}.fail"
    AST_OUTFILE="${TEST%.*}.ast.out"
    SAST_OUTFILE="${TEST%.*}.sast.out"

    if [ ! -f $OUTFILE ]; then
        echo "NO OUT" && echo "$RESULT" > $OUTFILE;
    else
        if [ "$RESULT" == "$(cat $OUTFILE)" ]; then
            echo "PASS $OUTFILE";
        else
            echo "FAIL $TEST differs from $OUTFILE";
            echo "OUT: " >> $FAILFILE
            echo $RESULT >> $FAILFILE; fi
    fi

    if [ ! -f $AST_OUTFILE ]; then
        echo "NO AST" && echo "$AST_RESULT" > $AST_OUTFILE
    else
        if [ "$AST_RESULT" == "$(cat $AST_OUTFILE)" ]; then
            echo "PASS $AST_OUTFILE";
        else
            echo "FAIL $TEST differs from $AST_OUTFILE";
            echo "AST: " >> $FAILFILE
            echo $AST_RESULT >> $FAILFILE; fi
    fi

    if [ ! -f $SAST_OUTFILE ]; then
        echo "NO SAST" && echo "$SAST_RESULT" > $SAST_OUTFILE
    else
```

```
        if [ "$SAST_RESULT" == "$(cat $SAST_OUTFILE)" ]; then
            echo "PASS $SAST_OUTFILE";
        else
            echo "FAIL $TEST differs from $SAST_OUTFILE";
            echo "SAST: " >> $FAILFILE
            echo $SAST_RESULT >> $FAILFILE; fi
    fi
done
exit
```

Testing Files

```
fun Int main() {
    Node<Int> a;
    Node<Int> b;

    a = '100';
    b = a;

    print(a.val);
    print(b.val);
    return 0;
}
fun Int main() {
    Bool a;
    a = Yes;
    if(a) {
        print("Bool works");
    }
    return 0;
}
fun Int main() {

        String a = "hi";

        Node<String> n = 'a';

        print("hi"[0]);
        print(a[0]);
        print(n.val[0]);

        return 0;
}
fun Int main() {
        Int b;

        b = 7;

        Int a = 5;
```

```
        Node<Int> start = '0';

        Graph<Int, String> g = <<
                start  -"h"-  '0'  -"i"-  '1',
                start  -"h"-  '0'  -"o"-  '1',
                start  -"s"-  '0'  -"u"-  start  -"p"-  '1'
        >>;

        print("Hell oWorld ");

        List<Edge<String> > adjacent = g.outgoing(start);

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

        return 0;
}
fun Int main() {
    Int a = 3;
    Int b;
    if( a== 3) {
        b = 5;
    }
    Int c = 35;
    print(b);
    print(c);
}
fun Int main() {

    Edge<String> aEdge;

    Node<Int> start = '0';
    Node<Int> state1 = '0';
    Node<Int> accept = '1';

    String input = "babbobababboba";
    Graph<Int, String> dfa = << start  -"b"-  state1  -"o"-  '0'  -"b"- '0'
    -"a"- accept, state1 -"a"- '0' -"b"-  accept -"b"- state1>>;
    Int token = 0;
    Int i = 0;
    Bool found = No;

    String character;
    List<Edge<String> > transitions;
```

```
    Node<Int> state = start;
    while( token < size(input)) {                       // String size

        character = input[token];                       // Get char at
    token
        transitions = dfa.outgoing(state);              // Get outgoing
    edges

        i = 0;
        while ((i < size(transitions)) and (found == No)) {
    // List size
            if (transitions[i].val == character) {      // Get list string
                state = transitions[i].to;              // Get edge
    destination
                found = Yes;
            }

            i = i + 1;
        }
        if ( found == No){
            print("REJECT");
            return 0;
        }
        found = No;
        token = token + 1;
    }

    if (state.val == 1) {               // Get accepting
        print("ACCEPT");
    } else {
        print("REJECT");
    }

    return 0;
}
fun Int main() {

        /// Grape Presentation:
                language : {h,i,a,b,c,o}
        ///

        Node<Bool> start = 'No';
        Node<Bool> reject = 'No';
        Node<Bool> accept = 'Yes';

        Graph<Bool,String> endB = <<
                start -"a"- start -"c"- start -"h"-
                start -"i"- start -"o"- start -"b"- accept,
                accept -"a"- start,
```

```
                accept -"c"- start,
                accept -"h"- start,
                accept -"i"- start,
                accept -"o"- start,
                accept -"b"- accept >>;
        print(dfa(endB, start, "hibob"));

        Graph<Bool,String> anyB = <<
                start -"a"- start -"c"- start -"h"-
                start -"i"- start -"o"- start -"b"-
                accept -"a"- accept -"b"- accept -"c"- accept -"h"-
                accept -"i"- accept -"o"- accept >>;
        print(dfa(anyB, start, "hibob"));

        Graph<Bool, String> greeting = <<
                start -"h"- 'No' -"i"-
                accept -"a"- accept -"b"- accept -"c"- accept -"h"-
                accept -"i"- accept -"o"- accept >>;

        String message = "histeve";

        Bool greetsBob = dfa(greeting, start, message) and
                         dfa(anyB, start, message) and
                         dfa(endB, start, message);

        if (greetsBob) { print("What's up bob!"); }
        else { print("Who are you?"); }
}
fun Int main() {
    Node<Bool> start = 'No';
    Node<Bool> oneA = 'No';
    Node<Bool> twoA = 'Yes';

    Graph<Bool, String> twoAs = <<
        start -"b"- start -"a"- oneA
        -"b"- oneA -"a"- twoA -"b"- twoA
        -"a"- 'No'
    >>;

    print(dfa(twoAs, start, "bbbbbabbb"));
}
fun Int dist(Graph<Int, Int> g, Node<Int> source, Node<Int> dest){
    List<Edge<Int> > outgo = g.outgoing(source);
    Int i = 0;
    while( i < size(outgo)){
        if(node_same(outgo[i].to, dest)){
            return outgo[i].val;
        }
        i = i+1;
```

```
    }
    return 100000;
}
fun List<Node<Int> > dijkstra( Graph<Int, Int> iter_graph, Node<Int>
   start_node, Node<Int> end_node){

        List<Node<Int> > visited = [];
        List<Node<Int> > result = [];
        List<Node<Int> > nodeList;
        List<Node<Int> > previous_record = [];
        List<Node<Int> > neighbor_list;
        Node<Int> next_node;
        Node<Int> dummy = '999';
        Int i;
        Int j;
        Int x;
        Int k;
        Int present;
        Int visSz;
        Int temp;
        Int distance;

        nodeList = graph_to_list(iter_graph);
        Node<Int> current_node = start_node;
        Int graph_size = size(nodeList);
        Int count = 0;

        //initializing previous_record with current node

        push_front_list_node(start_node, previous_record);
        while(count < graph_size-1){
                push_front_list_node(dummy, previous_record);
                count = count + 1;
        }

        //run until all nodes are visited

        while(size(visited) < graph_size and current_node.val != 85043210){
                neighbor_list = neighbor(current_node);
                i = size(neighbor_list);

                j = 0;

                //iter through neighbor list updating node value
                while (j < i){
                        present = 0;
                        k=0;
                        next_node = neighbor_list[j];
                        while(k < size(visited)){
```

```
                        if(node_same(visited[k], next_node)){
                            present = 1;
                        }

                        k = k + 1;
                    }
                    distance = dist(iter_graph, current_node,
next_node) + current_node.val;
                    //update previous record and distance
                    if (next_node.val > distance and present == 0){
                            next_node = update_node(next_node,
distance);

                            x = 0;

                            //update previous record
                            while(x < size(nodeList)){

if(node_same(nodeList[x] ,next_node)){
                                            previous_record =
update_at(current_node, previous_record, x);
                                    }
                                    x = x+1;
                            }
                    }
                    j = j+1;
            }

            //add_to visited and change current node
            push_front_list_node(current_node, visited);

            current_node = minNode(visited , nodeList);

    }


    Node<Int> follow = end_node;
    while( not(node_same(follow, start_node))){
            push_front_list_node(follow, result);
            temp = 0;
            while (temp < graph_size){
                    if(node_same(nodeList[temp], follow)){
                            follow = previous_record[temp];
                    }
            temp = temp + 1;
            }
    }

    push_front_list_node(start_node, result);
```

```
        return result;
}

fun Node<Int> minNode(List<Node<Int> > visited, List<Node<Int> > nodeList){

        Node<Int> target;

        Int min = 10000;
        Int j = 0;
        Int k;
        Int present;
        Bool found = No;
        while ( j < size(nodeList) ){

                k = 0;
                present = 0;

                while(k < size(visited)){
                        if(node_same(visited[k], nodeList[j])){
                            present = 1;
                        }

                        k = k + 1;
                }

                if (nodeList[j].val < min and present == 0){
                        target = nodeList[j];
                        min = nodeList[j].val;
                        found = Yes;
                }

                j = j + 1;
        }

        if(found == Yes) {
                return target;
        }
        return '85043210';
}

fun Int main(){

    Node<Int> a = '0';
    Node<Int> b = '10000';
    Node<Int> c = '10000';
    Node<Int> d = '10000';
    Node<Int> e = '10000';
    Node<Int> f = '10000';
    Node<Int> g = '10000';
```

```
    List<Node<Int> > nodes = [a,b,c,d,e,f,g];
    Graph<Int, Int> mainGraph = << a -2- b -5- d -3- g,
                                a -3- c -7- e -4- f -2- g,
                                d -2- f>>;

    Graph<Int, Int> mainGraph2 = << g -3- d -5- b -2- a,
                                g -2- f -4- e -7- c -3- a,
                                f -2- d>>;

    List<Node<Int> > result;

    result = dijkstra(mainGraph, a, g);
    print("Shortest Path:");
    Int i = 0;
    Int j = 0;
    Int reSz = size(result);

    while(i < reSz){
        j=0;
        while( j < size(nodes)){
            if(node_same(nodes[j], result[i])){
                if(j==0) { print("A"); }
                if(j==1) { print("B"); }
                if(j==2) { print("C"); }
                if(j==3) { print("D"); }
                if(j==4) { print("E"); }
                if(j==5) { print("F"); }
                if(j==6) { print("G"); }
                j = 10000;
            }
            j = j + 1;
        }
        i = i+1;
    }
    print("Total cost");
    print(result[reSz-1].val);
    return 0;
}


fun Int main() {

        Edge<Int> a = << -5-> >>;

        return 0;

}
fun Int main() {
```

```
        Edge<Int> a = <<-3->>;

        return 0;
}
fun Int main() {
    List<Int> a;
    a = [];
    return 0;
}
fun Int apple() {
    return peach();
}
fun Int peach() {
    return 3;
}
fun Int main() {
    print(apple());
    return 0;
}
fun Int foo(Int a) {
    return a + 5;
}

fun Int bar(Int a, Int b) {
    return a * b;
}

fun Int main() {
    Int a;
    Int b;
    a = 4;
    b = 1000;

    print(foo(a));
    print(bar(a,b));
}

fun Int main() {
    String a;
    a = "applejacks";
    print(a[5]);
    return 0;
}
fun Int main(){
    Node<Int> a = '5';
    Graph<Int, Int> graph = << a -6- '7' >>;
    List<Edge<Int> > nodeList = graph.outgoing(a);
    Edge<Int> edge= nodeList[0];
    Node<Int> toNode = edge.to;
```

```
    print(toNode.val);
    return 0;
}
fun Int main() {
        Graph<Int,Int> a;
        Node<Int> b;

        b = '3';

        a = << b -4- '5', b -10- '6' >>;

        return 0;
}
fun Int main() {
        Graph<Int, Int> a;

        a = <<'3' -3- '4'>>;

        return 0;
}
fun Int main() {
    Graph<Int, Int> a = << '7' -8- '9'>>;
    List<Node<Int > > b = graph_to_list(a);
    return 0;
}
fun Int main() { print("Hello World!"); }
fun Int main() {
        print("HI DUDE!");

        Int a = 5;

        print(a);

        String b = "Hi People!";

        print(b);

        return a;
}
fun Int main() {
        Int b;
        return 0;
}
fun Int main() {
    Int a = 2147483648;
    print(a);
    return 0;
}
fun Int main() {
```

```
        Node<Bool> start = 'No';
        Graph<Bool,String> aba = <<
                start -"a"- 'No' -"b"- 'No' -"a"- 'Yes' >>;
        if (dfa(aba, start, "aba")) {
                print("YAY");
        } else {
                print("nay");
        }
}
fun Int main() {
    List<Int> a = [1,2,3,4,5];
    print(a[5]);
    return 0;
}
fun Int main() {
    List<Edge<String> > a;
    Edge<String> b = <<-"hi"->>;
    a = [b];
    print(size(a));
    return 0;
}
fun Int main() {
    List<String> b;
    b = ["hi", "hello", "hello?" , "bye"];
    print(b[0]);
    return 0;
}
fun Int main() {
        List<Int> a = [1,2,3,4,5];

        print(a[3]);

        List<String> b = ["hi", "how", "are", "you"];

        List<Node<Int> > c = [
                '3', '4', '5', '6'
        ];

        print(b[2]);
}
fun Int main() {

        List<Node<Int> > a = ['3', '4'];

        print(a[0].val);

        return a[1].val;
}
fun Int main() {
```

```
        Node<String> start = '"James"';

        Graph<String, Int> = <<
                start -10-> '"India"' -20- '"Timmy"' -10-> '"Ed"',
                start -30-> '"Nick"' >>;

        print(start.friends)

}
fun Int main() {
        String hi = "hi";
        print(hi[0]);

        List<List<Int> > a = [
                [1,2,3,4,5],
                [1,2,3,4,5],
                [1,2,300,4,5],
                [1,2,3,4,5]];

        print(a[2][2]);

        List<List<Node<Int> > > b = [
                ['1','2','3','4','5'],
                ['1','2','3','4','5'],
                ['1','2','313','4','5'],
                ['1','2','3','4','5']];

        print(b[2][2].val);

        return 0;
}
fun Int main() {
        Int a = 3;

        Node<Int> b = 'a';
}
fun Int main() {
        Node<Int> a;
        a = '3';
        return 0;
}
fun Int main() {

        Int a = 3;
        Node<Int> b = 'a';
        Edge<String> e = << -"WASSUP"- >>;

        print(b.val);
```

```
        print(e.val);

        return 0;
}
fun Int main() {
    Graph<Int, Int> a;
    a = give_graph();
}

fun Graph<Int, Int> give_graph(){
    return << '5' -6- '7' >>;
}
fun Int main() {
        Int a = 3;
        Int b = 5;

        print(a+b);

        print(a * 3000);

        String z = "Hello";

        Graph<Int, String> g = << '3' -5- '6' >>;

        return 0;
}
fun Int main() {
    String a;
    a = "applejacks";

    print(size(a));
    return 0;
}
fun Int gime(Graph<Int, Int> a){
    return 5;
}

fun Int main() {
    Int a = gime(<<'5' -6- '7'>>);
    return a;
}
```

README

```
# The Grape Programming Language

### Development Environment

# Setting up environment
```

```
This could take a while, be patient!
```
$ ./run init
$ ./run dev
```

# Compilation (in VM)
```
$ ./run test
$ ./run build
```

### INFRA
- [X] Makefile
- [X] Dockerfile
- [ ] Test cases

### Scanner
- [X] List brackets
- [ ] Floats
- [X] Dictionary
- [X] modulo operator
- [X] exponent
- [X] ampersand "&" : graph union
- [X] for
- [X] in
- [X] Colon (for dictionary)
- [X] fun

### Parser
- [X] Implements List
- [ ] template searching (working for nodes not for edges, because edge notation conflicts with `MINUS expr`)
- [X] function declaration
- [X] dictionary declaration
- [ ] Edge declaration (shift/reduce with GT)


### Semant.mll
node, graph, type
- [ ] Don't assign types to empty lists, make new type: (List<>)
- [ ] Add types for graph
- [ ] Semant check if the last one has a no expr, if the node is not the end, there should be no expr.

### Codegen
- [ ] Link C library for lists
- [ ] Link C library for Node
- [ ] Link C library for Graph
```

C Library

Node
- [x] A list of edges
- [x] void pointer (data)

#dge
- [x] void(int) pointer (data)
- [x] node pointer (from)
- [x] ndoe pointer (to)

Graph
- [x] a list of node
- [x] function prototypes
- [ ] finish writing functions
- [ ] edge_init
- [ ] node_init
- [ ] graph_init:
1. Create a new graph
2. Add all nodes to graph
3. add all edges pointing to the node, and from the previous node
Then, for each ID:
3. Merge all graphs into one graph, resolving Nodes with that ID by removing all existing edges to that ID and pointing them to a new Node instance with that ID
5: return new graph

Suggestion:

LRM Changes:
- [ ] We can't have empty block stmt
- [ ] Dictionaries now used STR_LIT as keys