PRTZL Final Report

Mathew Mallett mm4673
Guanqi Luo gl2483
Rusty Nelson rnn2102

1.Introduction

We propose a language where vertices and edges are a first class data types. These types are used to construct, manipulate and test an implicit graph structure which is the main store in the language. A user of our language is able to create and manipulate graphs as easily as they can work with an array in most other languages. The language provides a way to construct the vertices and edges, as well as providing high level operations like finding neighbors and degrees of vertices. The data structures will be open enough to enable developers to build graph algorithms in the language as well as support a variety of different graph types from directed/undirected, weighted and restricted depending on the programmers adherence to their own rules.

2.Language Tutorial

PRTZL makes it easy to jump right into graph programming. In one line, you can make a vertex. In 3, you can have the start of a graph with 2 vertices linked by an edge. Let's get started. Create a new file and add this line:

```
Vertex houston = <+ “Houston” +>;
```

We've just create a vertex with name “Houston”! Now let's create another edge and link them together.

```
Vertex austin = <+ “Austin” +>;
link(houston, austin, 1);
```

Now we have 2 vertices, Houston and Austin, and they have an edge between them, from houston to austin. Say we want to follow that edge, that's easy as well.

```
List out_list;
out_list = houston.out;
Edge e;
e = out_list[0];
Vertex dest = e.dest;
```

That looks kind of busy, let's break it down. We access the out property of the Houston vertex, which is a List of all outbound edges for Houston. We grab the first (and only) edge in that list, then access the dest property to get the city, Austin which is stored at the other end of the edge.

We've shown how to create vertices and edges, and how to walk a simple graph. What if we need to store some information about a vertex? Simple, store it in a property.

```
houston.population = 2196000;
```

And to get that information later,

```
houston.population;
```

Along with that, we have if statements and while loops:

```
if(condition)
      /* conditional code */
endif

while(condition) do
      /* loop body */
endwhile
```

PRTZL has 5 core types: Number, String, Vertex, Edge, List. You can declare a variable of the given type by using the type as a modifier. Below shows declaration and initialization of each of the types. Numbers are backed by floating point numbers, and also serve as booleans. A 0 is false, anything else is true. Vertexes and Edges are created through interaction with the graph.

```
Number mynumber = 3.1;
String mystring = “hello”;
Vertex myvertex = <+ mystring +>
Edge myedge; /* only created through link() */
List mylist = [];
```

With these building blocks, you should have enough to get started creating more complex graph algorithms. For a more complete tour of what PRTZL has to offer, see our language reference manual section below.

Compiling

Once you have written your PRTZL source, pipe it into the compile binary. This will output C code, which you can compile with GCC. Be sure to link our C libraries to GCC.

```
./compile < mycode.prtzl > out.c
gcc out.c prtzl.c map.c list.c
```

3.Language Reference Manual

1.Lexical Conventions

There are six types of tokens: identifiers, keywords, number literals, string literals, operators, and other separators. Whitespace is ignored, unless it is included within a string literal.

Reserved Keywords

The following keywords are reserved by the language and not available for variable or function names

```
Number      String          Vertex          Edge
if          elseif          else            endif
while       do              endwhile
return      endfunc         null
```

Comments

```
/*  this is a comment */
/* also
a
comment */
```

Identifiers

An identifier is an alphanumeric string. The first character must be alphabetic. We also allow _ within identifiers. Identifiers are case sensitive.

Separators

We separate statements with ‘;’. Anything after a ‘;’ is a new statement.

2. Types

We have 5 types

Number

- Can represent both integer and floating point numbers.
- The number 0 is considered “falsey” as a boolean value, all other Number values are “truthy”.
- Numbers take a default value of 0.
- Numbers are mutable. You are able to reuse variables and rewrite the value stored within them multiple times.

String

- Represents a string of characters.
- Strings take a default value of “” (empty string)
- Strings are immutable. Rewriting a String results in the creation of a new String buffer.

Vertex

- Represents a vertex within the graph.
- Vertex variables take a value of **null** until they are bound to a Vertex within the graph.
- Vertexes are mutable. You are able to write changes to Vertex variables.
- Vertexes have a key/value attribute system attached to them. You are able to attach additional information about a specific vertex through a key/value pair.

Edge

- Represents an edge between two Vertexes.
- Edge variables take a value of **null** until they are bound to an Edge within the graph.
- Edges are mutable. You are able to write changes to Edge variables.
- Edges have a key/value attribute system attached to them. You are able to attach additional information about a specific vertex through a key/value pair.

List

- A list containing any of the other data types. Each list is able to hold one data type. Lists are mutable, you can add and remove items from a list.
- Lists variables are initialized to **null**

null

null is a keyword that indicates that a variable has not had a valid pointer assigned to it, or otherwise does not point to a valid object.

Literals

We have number and string literals, as well as an empty list literal.

Number literal

```
1
42
3.141592
```

String literal

```
“I am literally a literal”
“Hello, World!”
“”
```

Empty List literal

```
[]
```

3.Expressions

Below we outline the expressions within our language. We also define order of operations and left or right associativity for each operation. An expression is made up of one or more of the operators listed below. You are able to naturally combine operators to form sequences of complex operations, as long as the types align correctly.

Variable declaration

A declaration is made of two tokens: a type identifier, and a variable identifier. Once a variable identifier has been used within the current scope, you aren’t able to use that identifier again. If you reuse an identifier within a more specific scope, the identifier of the more specific scope will shadow the less specific identifier, just as in C. Variable declarations must be the leftmost section of an expression.

Here are declarations of our 5 types:

```
Number my_num;
String my_string;
Vertex my_vert;
Edge my_edge;
List my_list;
```

Assignment

Assign a literal or another expression to a variable

```
my_num = 3;
my_string = “I am literally a literal”;
```

You can declare and assign in the same expression

```
Number my_other_num = 3.141592;
my_num = my_other num + 32;

Vertex my_other_vert = my_vert;
```

Assignment is right associate. Everything to the right of = is evaluated first. The left side of = (lvalue) must be either a variable declaration or a variable identifier. Other expressions are not valid lvalues.

Binary operators

We support standard operators on Number type, with PEMDAS order of operations

```
+
-
*
/
```

```
my_num = 1 + 2 / 3;
Number my_other_num = my_num + 3.141592;
```

You can concatenate String types with the ^

```
String my_string = “I am literally “ ^ “a literal”;
String my_longer_string = my_string ^ “ Hello, World!”;
```

Binary operators are left associative. The left side of the expression is evaluated first, then the right side, then the operator is applied.

Parenthesis

Use parenthesis to increase the precedence of a sub-expression

```
Number my_number = (1 + 2) * 3;  /* 9 */
```

Binary comparators

We can compare two numbers with the C style comparators

==
!=
>
>=
<
<=

Comparators return 1 for true, 0 false

```
Number my_num = 3.14 > 3.13; /* 1 */
my_num = 24 <= -24; /* 0 */
```

Comparators are left associative.

Unary operators

Unary minus (-) inverts the sign on a Number

```
Number my_number = 42;
my_number = -my_number; /* -42 */
```

Not operator inverts the logical truth value of the Number. 0 is false, anything else is true

```
Number my_number = !0; /* 1 */
my_number = !(1 + 1); /* 0 */
```

Precedence of operators

In order of most to least precedence

! - (Unary minus)
/ *
+ -
== != < <= > >=
=

Lists

Allow you to store lists of elements, where all contained elements are the same type. You can create a list with the empty list literal []. Add elements to the list by storing items into the index. You can get a specific index with the [] syntax similar to C, you can get the length with list_length().

```
List my_list = [];
my_list[0] = 3.14;
my_list[1] = 42;

my_list[0]; /* 3.14 */

list_length(my_list); /* 2 */
```

3.1.Graph Operators

There are three primary operations that interface with the graph. The graph is implicitly declared in any PRTZL program, and we provide operators to insert, retrieve, and remove vertices from it.

Insert key operator <+ String string +>

Inserts the string key into the graph. String can be a variable or a literal.

```
<+ “Detroit” +>; /* insert detroit vertex into the graph */

String my_string = “Houston”
<+ my_string +>;
```

Query key operator <? String string ?>

Check the existence of a vertex within a graph, returns the Vertex object, or null object

```
Vertex my_vertex = <? “Detroit” ?>; /* gives the Vertex for Detroit */
Vertex my_other_vertex = <? “Miami” ?>; /* gives null */

my_other_vertex == null; /* gives 1 */
```

Delete vertex from the graph <- String string ->

Attempts to delete the vertex from the graph. If it does not exist, returns 0, if it does exist, it removes all edges to and from the vertex, then removes it from the graph structure, returning 1

```
Number success = <- “Detroit” ->; /* Detroit is deleted, and returns true */
success = <- “Detroit” ->; /* Detroit is not in the graph this time, now returns false; */
```

Vertex data fields

You can store key/value pairs in a Vertex object to associated data with that Vertex with the . operator. Gives null on properties that have not been set yet. You can put Number, String, Vertex, Edge, and Lists into a value. Keys are alphanumeric and allow _ .

```
Vertex houston = <? “Houston” ?>;
houston.visited = 0;
houston.visited; /* 0 */

houston.other; /* null */
```

All Vertices have the following properties built into them

`out`	List of outbound Edges
`in`	List of inbound Edges
`in_degree`	Number of inbound edges
`out_degree`	Number of outbound edges

Edges

Edges are similar to Vertices. They are created when you link two Vertices together. You are able to attach key/value pairs to Edges in the same manner as Vertices.

All Edges have the following properties in their key/value map.

`to`	Destination Vertex
`from`	Source Vertex
`weight`	Number weight of the edge, if not specified, is 1

4.Control Flow

Conditional branching

Use if elseif else to group conditional branches together. Each conditional flow has an **if**, and optionally has more conditionally executed statements as **elseif**. **elseif** is executed if the preceding **if** and **elseif**'s did not evaluate to true. If none of the conditions were true, **else** clause is invoked. The end of a string of **if elseif else** is marked with an **endif**. Every conditional flow is marked by one **if**/**endif** pair, optionally many **elseif**, and optionally one **else** as the final conditional branch.

```
if(1 == 0)
      /* do this */
elseif(1 == 2)
      /* do this */
elseif(1 == 3)
      /* do this */
endelseif
else
      /* do this */
endif

/* another example */
if(1)
      /* conditional body */
endif
```

Loops

We support the **while** loop. The for loop has 2 pieces. The first is a statement that determines the number of iterations of the loop, the second is the body of the loop. Anything declared in the for statement has loop scope, not global scope. The variables exist only within the loop and are reclaimed after the loop finishes.

Broken down:

```
while (/* continue looping condition */) do
      /* loop body */
end
```

And an example iterating a List.

```
List my_list = [];
/* list is filled here */

Number i = 0
while( i < list_length(my_list)) do
      my_list[i]; /* do something with this element */

      i = i + 1;
endwhile
```

5.Functions

Function declaration is similar to C, with a type, label, and argument signature. The end of the function is denoted with the **end** keyword.

A breakdown of the function declaration with optional pieces marked with [ ]

```
return_type function_name([arg_type arg_name, arg_type arg_name])
      [ function body, has function scope ]
      return a_variable_of_return_type;
end
```

/* here is a demonstration */

```
Number my_function(Number arg1, Number arg2)
      /*  variables declared here have function, not global, scope */
      Number local_var = 2;
      return arg1 + arg2 + arg3;
endfunc
```

Functions can be recursive. Functions must be declared within global scope.

Calling a function

```
my_function(arg1, arg2);
```

6.Scope Rules

Scoping is simple. We have global scope, if/else scope, loop scope, and function scope.

Anything not declared within an if/else statement, loop, or function is global scope. Globally scoped identifiers are visible anywhere within the program.

If/else, loop, and function scope all work the same. Anything declared within the bodies of these structures have scope limited to the body of the structure. You are able to reuse identifiers that have been used at global scope, the more specific scope will shadow the global identifier. Anything declared within the more specific scope will go away after the body finished (if completes, loop finishes, function returns).

A nested if/else or loop will have its own scope, the same rules apply. You are able to see everything from the higher level scope, but the higher level scope doesn't have access to the more limited scope.

7.Standard Library

The following function are built into the language and available to use at any time.

Number link(**Vertex** a, **Vertex** b, **Number** w)

Creates a unidirectional Edge link from Vertex a to Vertex b with weight w
Returns 1 on success

Number bi_link(**Vertex** a, **Vertex** b, **Number** w)

Create a bidirectional Edge link from Vertex b to Vertex b and back with weight w
Internally, creates 2 Edges
Returns 1 on success

Number add(**List** l, <**Type**> item)

Add a new entry to the end of the list. Type must match the type of the list.
Returns 1 on success

Number remove(**List** l, **Number** index)

Removes the index from the list
Returns 1 on success

Number list_length(**List** l)

Returns the length of the list

Number string_length(**String** s)

Returns the length of the String

Number print_number(**Number** n)

Prints the number to STDOUT

Number print_string(**String** s)

Prints the string to STDOUT

Number print_vertex(**Vertex** v)

Prints the Vertex to STDOUT

Number print_edge(**Edge** e)

Prints the Edge to STDOUT

Number cmp(**String** a, **String** b)

Checks for equivalence of a and b
Returns 1 if equivalent

4.Project Plan

Process

Ours is a distributed team. Two of our team members work full time, so we have to work remotely and communicate online. This creates additional challenges for an already tight deadline, but we've leveraged a number of tools to help us. We meet as a team twice a week, directly after class. In this meeting, we discuss what we accomplished since the last meeting, and go over what we want to accomplish for our next meeting. We also meet over Skype for working sessions in between our class meetings, to keep everyone on the same page and work through issues together.

We use Github for source control and project planning. We track our work items with the Github issues tool, and keep the team updated on progress through the tool and through email. We use Github plus Travis CI for continuous testing of our code. Every commit to our master branch is automatically run through our testing suite, and the team receives email updates on the success or failure of the commit.

Programming Style Guide

PRTZL consists of a mixture of OCAML, C, and PRTZL source files. Each section has it's own style guide, which we will outline breifly.

C

PRTZL C code follows standard C conventions. Variables are lower case and use underscore to separate words in identifiers. Functions follow the same convention. Declarators are on the same line as their identifiers.

```
struct node* new_vert;
struct node* insert_vertex(struct graph* g, char* label){
```

Sections of code enclosed by a set of curly brackets gain one level of indentation. Indentation is provided through an additional tab. The opening curly bracket is on the end of the opening statement of the enclosure. The closing bracket is on a line by itself, at the same level of indentation as the opening statement.

```
struct node* insert_vertex(struct graph* g, char* label){

      struct node* new_vert = _init_vertex(label);
      map_put(g->vertices, label, new_vert);
      return new_vert;
}
```

All function and struct definitions are placed in header files, and the implementation of the functions exclusively are in the .c source files.

Ocaml

Ocaml code follows the standard conventions as presented by Dr. Edwards. Variables and functions follow the same style guidelines as our C code, with lower case, underscore separated identifiers. Abstract data types are capitalized, as an exception.

```
let string_of_datatype = function
type datatype = Number | String | Vertex | Edge | Void | List
```

We follow a similar indentation rule to C, where enclosed statements gain an additional level of indentation.

```
let string_of_datatype = function
    Number -> "Number"
  | String -> "String"
  | Vertex -> "Vertex"
  | Edge   -> "Edge"
  | Void   -> "Void"
  | List   -> "List"
```

PRTZL

PRTZL source is similar in syntax to C. It follows the same style guidelines as C, with the difference that we replace curly brackets with end identifiers (example - `endif`).

Project Timeline

6/3/15	Proposal completed
6/15/15	Language Reference Manual completed
6/15/15	Lexer and Parser completed
6/18/15	“Hello World” compiler
6/28/15	Majority of development completed
7/2/15	Report completed, presentations

Member Roles

Mathew Mallett	Language Guru C libraries Documentation
Guanqi Luo	System Architect Core compiler architecture and implementation
Rusty Nelson	Manager, Verification & Validation Github and Travis CI build system Test Automation

Project Development Environment

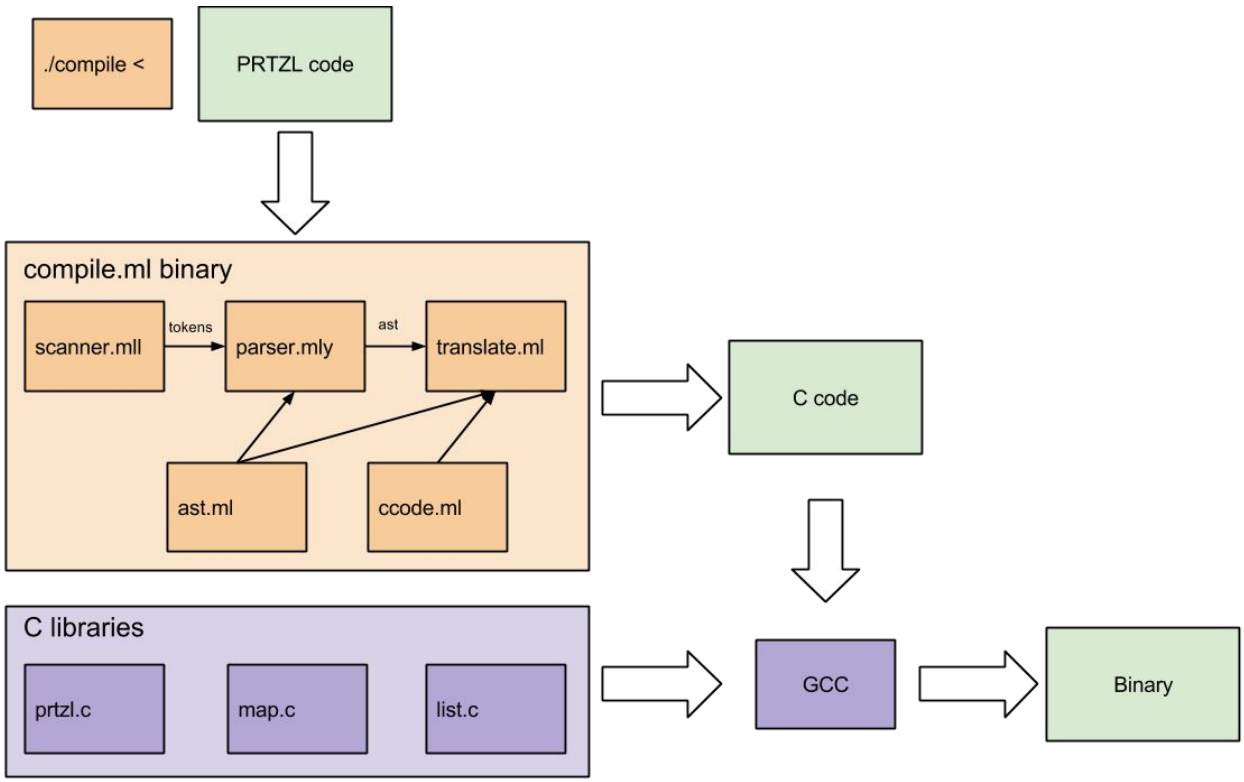
Development is done in a Linux environment. Continuous build is done on Ubuntu 12.04 provided by Travis CI. The project is written in a combination of Ocaml and platform independent C. We target version 4.01.2 of Ocaml. For builds we use the make command line tool, coupled with a makefile, to build our Ocaml source. Our generated C code is simple and is built with GCC with a single command. We use git for version control, using Github.com as the repo provider. We also leverage Github’s issues and email for passing updates to the team. We use Skype and email for team collaboration.

GitHub: https://github.com/doctorrustynelson/prtzl
Travis-CI: https://travis-ci.org/doctorrustynelson/prtzl

Project Log

6/3/15	Proposal completed
6/15/15	Language Reference Manual completed
6/15/15	Lexer and Parser completed
6/18/15	“Hello World” compiler
6/24/15	String, Number operations, if statement, functions
6/26/15	while loop statement
6/28/15	Type checking, graph operations
7/2/15	Report completed, presentations

5.Architectural Design



PRTZL has 2 main components. The OCAML compiler takes PRTZL source and generates C code. The C code is then linked with PRTZL c libraries for the backing implementation of graph data structures, and compiled into a binary.

The entry point of the compiler is compile.ml. This file runs in series the lexer, parser, and translator portions of the compiler.

scanner.mll is an Ocamllex program that takes a PRTZL source file and tokenizes it. The tokens are then passed on to parser.mly.

parser.mly is an Ocamlyacc program that matches sequences of tokens to statements and expressions in PRTZL, as they are defined in ast.ml. Once the token sequences have been matched to language constructs, they are passed to translate.ml.

translate.ml takes the output of parser.mly and recursively expands them into C constructs, as they are defined in ccode.ml. The C code constructs are then expanded into a string of C code, which is output.

Once the C code has been generated, it can be compiled with GCC. We link in the libraries that define and implement PRTZL data structures through the PRTZL, map and list source and header files. We compile all of the files into a single binary with GCC that can then be run just as any other compiled C program.

Division of work

Most of the Ocaml compiler is written by Kay (Guanqi), with contributions by Matt.
The C libraries are written by Matt.
The test suite and test programs are built by Rusty.

6.Test Plan

Test Suite Functional Overview

Empty	empty program is valid
Comments	comments are processed correctly
Functions	functions can be defined and called
Graph_Ops	graph insert, query, and delete operators work
Hello	hello world base test
If	if statements execute correctly
Link	linking vertices produces expected behavior
List	list type works correctly
Numbers	numerical operators work correctly
Print	print functions work
Properties	Vertex and Edge property map works correctly
While	while statement works correctly
Strings	String type operators work as expected

Criteria of Choosing Tests

We wrote at least one test case for every functional piece (shown above) of our language most of the time we had multiple test cases. If there was an entry in our Language Reference Manual, we wrote at least one test to verify that it works. We are verifying that our language both compiles correctly, and that it generates the expected output. Each test case has 3 phases. In the first phase, our Ocaml compiler attempts to compile the .prtzl source file into C code. Next, we attempt to compile the C code with GCC, linking the PRTZL C libraries. Finally, we run the generated binary and compare it against an expected output. If all three phases pass, that test case passes.

There were 30 test cases that we constantly used and submitted into the core project.

Automation

We use Ruby's rake task runner to automate our tests. The Rake job has three major tasks. The first task does a fresh compile of our Ocaml compiler. The second tasks sets up the test environment and runs the test suite. It runs each .prtzl file through the compiler, then uses gcc to compile the generated c, and then runs the binary and uses diff to compare it with an expected output. The final task cleans up all of the generated files, so the next test will have a clean slate.

We have set up Travis-CI for continuous integration. Every time we pushed a commit to GitHub or submitted a pull request, our test suite is run, and sends emails letting us know whether the tests passed or failed.

Division of Work

Cooperatively

- Parser and Scanner
- Demo

Matt

- Rake job
- .prtzl and .out files for test suite

Rusty

- Travis-CI build system setup
- GitHub project management

Guanqi

- Compiler architecture
- Translation of PRTZL to c

Sample Translated Programs

graph_ops.prtzl

```
String houston = "houston";
String dallas = "dallas";

Vertex v = <+ houston +>;
Vertex c = <+ "austin" +>;
<? dallas ?>;
<- houston ->;
```

graph_ops_test.c

```
#include "prtzl.h"
struct graph* g;
int main() {
g=init_graph();
char* houston = "houston";char* dallas = "dallas";struct node* v =
insert_vertex(g, houston);struct node* c = insert_vertex(g,
"austin");query_vertex(g, dallas);delete_vertex(g, houston);
       return 0;
}
```

functions.prtzl

```
Number one()
      return 1;
endfunc

Number add_one(Number x)
      return x + one();
endfunc

Number z = 3.1;
Number y = add_one(z);
Number w = add_one(3.1415);

String useful_function(String a, String b)
      return a ^ b;
endfunc

String hello = "hello ";
String world = "world";

String hello_world = useful_function(hello, world);
```

functions_test.c

```
#include "prtzl.h"
struct graph* g;
double one()
{
return 1;
}
double add_one(double x)
{
return x + one();
}
char* useful_function(char* a, char* b)
{
return cat(a, b);
}
int main() {
g=init_graph();
double z = 3.1;double y = add_one(z);double w = add_one(3.1415);char* hello
= "hello ";char* world = "world";char* hello_world =
useful_function(hello,world);
       return 0;
}
```

Sample Test Output

```
vagrant@athena:/vagrant/prtzl/test$ rake test
(in /vagrant/prtzl)
Running tests

comments
      Compiling .prtzl  OK
      Compiling .c      OK
      Running .o        OK
empty
      Compiling .prtzl  OK
      Compiling .c      OK
      Running .o        OK
functions
      Compiling .prtzl  OK
      Compiling .c      OK
      Running .o        OK
graph_ops
      Compiling .prtzl  OK
      Compiling .c      OK
      Running .o        OK
hello
      Compiling .prtzl  OK
      Compiling .c      OK
      Running .o        OK
if
      Compiling .prtzl  OK
      Compiling .c      OK
      Running .o        OK

      . . .

while
      Compiling .prtzlFatal error: exception Parsing.Parse_error
      FAILED
      Compiling
.c/usr/lib/gcc/x86_64-linux-gnu/4.6/../../../x86_64-linux-gnu/crt1.o: In
function `_start':
(.text+0x20): undefined reference to `main'
collect2: ld returned 1 exit status
            FAILED
      Running .osh: 1: ./while.o: not found
            FAILED
```

7.Lessons Learned

Matt

I found it interesting and rewarding to see how a whole language comes together. I've written templating systems before, but an entire language is much more complex and intricate. As Dr. Edwards stated many times this semester, the real challenge is in all the corner cases.

My word of advice to later students: It seems like a lot of work, but it is definitely possible. The first version of JavaScript was completed in 10 days, and your language will be much simpler than JavaScript

Kay (Guanqi)

I used to think that theory and practice are loosely bound because it's hard for me to relate use cases in practice to theories. While doing this project, I found it amazing how theory and practice are closely related in implementing a compiler and the transition from theory to practice is very clear. Better understanding of the theories is helpful to implementation.

My word of advice to later students: Get the simplest functionality done first and start testing as soon as possible because there are numerous ways a small functionality could fail and debugging is not easy.

Rusty

If I could do it all again the first thing I would have done is have everyone sit down and evaluate the test and development environment. We went though a few iterations of the test infrastructure for a variety of reasons. Sitting down and ironing out those issues at the beginning of the project would have saved us quite a bit of time later on. It would have also made it easier to get our compiler up and running.

In addition to getting things up and running early on I would have like it if we could have set up intermediary testing between the compilation steps to better verify our system. This I think would also have greatly increased our productivity by providing greater confidence in our previous steps as well as aided in the tracking down of bugs.

My word of advice to later students: Get the testing environment sorted out so that as soon as you have hello world working you can start setting up tests which should help in figuring out future work.

8.Appendix

src/ast.ml

```
type operator = Add | Sub | Mul | Div | Equal | Neq | Less | Leq
                            | Greater | Geq | Concat

type datatype = Number | String | Vertex | Edge | Void | List


type expr =
      Num of float
    | Str of string
    | Id of string
    | Int of int
    | Assign of string * expr
    | Binop of expr * operator * expr
    | Not of expr
    | Neg of expr
    | Insert of expr
    | Delete of expr
    | Query of expr
    | List of expr list
    | Mem of string * expr
    | ListAssign of string * expr * expr
    | Call of string * expr list
    | Keyword of string
    | Vertex of string
    | Edge of string
    | Property of string * string
    | PropertyAssign of string * string * expr
    | AddParen of expr
    | Noexpr



type stmt =
            Block of stmt list
          | Expr of expr
          | If of expr * stmt list * stmt list * stmt list
          | Elseif of expr * stmt list
          | While of expr * stmt list
          | Return of expr
```

```
type vdecl = {
        vtype : datatype;
        vname : string;
        value : expr;
}

type formal = {
        ftype : datatype;
        frname : string;
}

type func_decl = {
        rtype : datatype;
                fname : string; (* Name of the function *)
                formals : formal list; (* Formal argument names *)
                locals : vdecl list; (* Locally defined variables *)
                body : stmt list;
}

type program = vdecl list * stmt list * func_decl list
```

src/ccode.ml

```
type cstmt =
          Strg of string
|         Numb of float
|         Int        of int
|         Id         of string
|   Vertex of string
|         Edge of string
|         Keyword of string
|   Main
|         Endmain
|         Datatype of Ast.datatype
|         Binop   of (string * cstmt list) * Ast.operator * (string * cstmt list)
|   Assign of string * (string * cstmt list)
|         Not of cstmt list
|         Neg of cstmt list
|         Call of string * cstmt list
|   List of string * cstmt list
|   Mem of string * cstmt list
|         ListAssign of string * cstmt list * cstmt list
```

```
|       If of cstmt list
|       Then of cstmt list
|       Else of cstmt list
|  Elseif of cstmt list * cstmt list
|  While of cstmt list * cstmt list
|  Return of cstmt list
|       Insert of cstmt list
|  Query of cstmt list
|       Delete of cstmt list
|       Property of string * string
|       PropertyAssign of string * string * cstmt list
|       AddParen of cstmt list

type prog = {
        text : cstmt array;
}
```

src/compile.ml

```
let _ =
  let lexbuf = Lexing.from_channel stdin in
  let program = Parser.program Scanner.token lexbuf in
  (*Translate.translate program*)
  let result = Translate.translate program in
    (List.iter (fun x -> print_string (Translate.string_of_ccode ("", x) )) result)
```

src/eval.ml

```
open Ast

let ary = Array.make 10 "0"
(*let hash = Hashtbl.create 100*)



  let rec eval = function
      Num(x) -> print_string " NUM "; string_of_float x
    | Str(x) -> print_string " STR "; x
    | Id(x) -> print_string " ID "; x
    | Int(x) -> print_string " Int "; string_of_int x
    | Neg(x) -> string_of_float (-. float_of_string (eval x) )
    | Not(x) -> let v1 = eval x in
                if (float_of_string v1) > 0. then string_of_int 0 else string_of_int 1
    | Assign(e1, e2) ->
```

```
    let v1 = e1 and v2 = eval e2 in
    v1 ^ " Assigned " ^ v2 (*ary.(v1 - 1) <- v2; ary.(v1-1) *)
| List(e1) -> List.iter (fun x -> print_string (eval x) ) e1; " list"
| Mem(e1, e2) -> "Mem"
| Insert(e1) ->
    let v1 = eval e1 in
    print_string "inserting "; v1
| Delete(e1) ->
    let v1 = eval e1 in
    print_string "deleting "; v1
| Query(e1) ->
    let v1 = eval e1 in
    print_string "querying ";  v1
| Call(e1, e2) -> "call"
| Binop(e1, op, e2) ->
    let v1 = eval e1 and v2 = eval e2 in
    match op with
            Add -> string_of_float (float_of_string (v1) +. float_of_string (v2))
    | Sub -> string_of_float (float_of_string (v1) -. float_of_string (v2))
    | Mul -> string_of_float (float_of_string (v1) *. float_of_string (v2))
    | Div -> string_of_float (float_of_string (v1) /. float_of_string (v2))
    | Equal -> string_of_int ( if v1 = v2 then 1 else 0 )
    | Neq -> string_of_int ( if v1 = v2 then 0 else 1 )
    | Less -> string_of_int ( if v1 < v2 then 1 else 0 )
    | Leq -> string_of_int ( if v1 <= v2 then 1 else 0 )
    | Greater -> string_of_int ( if v1 > v2 then 1 else 0 )
    | Geq -> string_of_int ( if v1 >= v2 then 1 else 0 )
    | Cancat -> v1 ^ v2

let rec exec = function
      Expr(e) -> print_string (eval e); "Expr "
    | Block(stmts) -> List.iter (fun x -> print_string "executing ") stmts; "Block"
    | If(e, s1, s2, s3) ->
      let v = int_of_string (eval e) in
      exec (if v != 0 then s1 else s3)
        | While(e, s1) ->
      let v = int_of_string (eval e) in
      (if v != 0 then exec s1 else "skipped")
    (*| Elseif(e, s1) ->
      let v = int_of_string (eval e) in
      exec (if v != 0 then s1)*)
```

```
  let _ =
    let lexbuf = Lexing.from_channel stdin in
    let stmt = Parser.stmt Scanner.token lexbuf in
    let result = exec stmt in
    print_endline (result)
```

src/Makefile

```
OBJS = ast.cmo parser.cmo scanner.cmo ccode.cmo translate.cmo compile.cmo

TESTS = \
arith1 \
arith2 \
fib \
for1 \
func1 \
func2 \
func3 \
gcd \
global1 \
hello \
if1 \
if2 \
if3 \
if4 \
ops1 \
var1 \
while1

TARFILES = Makefile testall.sh scanner.mll parser.mly \
	ast.ml ccode.ml translate.ml compile.ml \
	$(TESTS:%=tests/test-%.mc) \
	$(TESTS:%=tests/test-%.out)

compile : $(OBJS)
	ocamlc -o compile $(OBJS)

.PHONY : test
test : compile testall.sh
	./testall.sh

scanner.ml : scanner.mll
	ocamllex scanner.mll
```

```
parser.ml parser.mli : parser.mly
	ocamlyacc parser.mly

%.cmo : %.ml
	ocamlc -c $<

%.cmi : %.mli
	ocamlc -c $<

compile.tar.gz : $(TARFILES)
	cd .. && tar czf compile/compile.tar.gz $(TARFILES:%=compile/%)

.PHONY : clean
clean :
	rm -f compile parser.ml parser.mli scanner.ml testall.log \
	*.cmo *.cmi *.out *.diff

# Generated by ocamldep *.ml *.mli
ast.cmo:
ast.cmx:
ccode.cmo: ast.cmo
ccode.cmx: ast.cmx
translate.cmo: ccode.cmo ast.cmo
translate.cmx: ccode.cmx ast.cmx
compile.cmo: scanner.cmo parser.cmi translate.cmo \
   ccode.cmo ast.cmo
compile.cmx: scanner.cmx parser.cmx translate.cmx \
   ccode.cmx ast.cmx
parser.cmo: ast.cmo parser.cmi
parser.cmx: ast.cmx parser.cmi
scanner.cmo: parser.cmi
scanner.cmx: parser.cmx
parser.cmi: ast.cmo
```

src/parser.mly

```
%{ open Ast %}

%token PLUS MINUS TIMES DIVIDE EQ NEQ GREATER LESS GEQ LEQ ASSIGN QUOTE
CONCAT DOTOPT NOT COMMA SEMI
%token LPAREN RPAREN LINSERT RINSERT LDELETE RDELETE LQUERY RQUERY
LBRACKET RBRACKET
```

```
%token NUMBER STRING VERTEX EDGE LIST IF ELSE ELSEIF ENDIF ENDELSEIF WHILE
DO ENDWHILE ENDFUNC RETURN EOF
%token <float> LITERAL
%token <string> ID
%token <string> STR
%token <int> INT

%nonassoc NOELSE /* Precedence and associativity of each operator */
%nonassoc ELSE
%right ASSIGN
%left EQ NEQ GREATER LESS GEQ LEQ
%left PLUS MINUS
%left TIMES DIVIDE CONCAT
%nonassoc UMINUS NOT

%start program
%type < Ast.program> program

%%

program:
  /* nothing */   { [], [], [] }
| program vdecl { let (a,b,c) = $1 in ($2 :: a), b, c }
| program stmt { let (a,b,c) = $1 in a, ($2 :: b), c }
| program fdecl { let (a,b,c) = $1 in a, b, ($2 :: c) }


fdecl:
  NUMBER ID LPAREN arguement_list RPAREN vdecl_list stmt_list ENDFUNC
          { {
      rtype   = Number;
                  fname   = $2;
          formals = List.rev $4;
          locals  = List.rev $6;
          body    = List.rev $7 } }
| STRING ID LPAREN arguement_list RPAREN vdecl_list stmt_list ENDFUNC
          { {
      rtype   = String;
                  fname   = $2;
          formals = List.rev $4;
          locals  = List.rev $6;
          body    = List.rev $7 } }
| VERTEX ID LPAREN arguement_list RPAREN vdecl_list stmt_list ENDFUNC
```

```
        { {
    rtype   = Vertex;
                fname   = $2;
        formals = List.rev $4;
        locals  = List.rev $6;
        body    = List.rev $7 } }
| EDGE ID LPAREN arguement_list RPAREN vdecl_list stmt_list ENDFUNC
        { {
    rtype   = Edge;
                fname   = $2;
        formals = List.rev $4;
        locals  = List.rev $6;
        body    = List.rev $7 } }

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

vdecl:
  NUMBER ID SEMI    { {vtype=Number; vname=$2; value=Assign($2, Num(0.))} }
| STRING ID SEMI    { {vtype=String; vname=$2; value=Assign($2, Str("\"\"") )} }
| VERTEX ID SEMI    { {vtype=Vertex; vname=$2; value=Assign($2, Vertex("") )} }
| EDGE ID   SEMI    { {vtype=  Edge; vname=$2; value=Assign($2, Edge("") )} }
| LIST ID   SEMI    { {vtype=  List; vname=$2; value=Assign($2, List([]))} }
| NUMBER ID ASSIGN expr SEMI { {vtype=Number; vname=$2; value=Assign($2, $4)} }
| STRING ID ASSIGN expr SEMI { {vtype=String; vname=$2; value=Assign($2, $4)} }
| VERTEX ID ASSIGN expr SEMI { {vtype=Vertex; vname=$2; value=Assign($2, $4)} }
| EDGE   ID ASSIGN expr SEMI { {vtype=  Edge; vname=$2; value=Assign($2, $4)} }
| LIST   ID ASSIGN expr SEMI { {vtype=  List; vname=$2; value=Assign($2, $4)} }

arguement_list:
              { [] }
| arguement     { [$1] }
| arguement_list COMMA arguement            { $3 :: $1 }

arguement:
  NUMBER ID     { {ftype = Number; frname = $2} }
| STRING ID     { {ftype = String; frname = $2} }
| VERTEX ID     { {ftype = Vertex; frname = $2} }
| EDGE ID       { {ftype = Edge; frname = $2} }
| LIST ID       { {ftype = List; frname = $2} }

stmt_list:
```

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

elseif_list:                                                              /* rules never
reduced*/
  /*nothing*/                    { [] }
| elseif_list elseif        { $2 :: $1 }

elseif:
  ELSEIF LPAREN expr RPAREN stmt_list   { Elseif($3, List.rev $5) }

stmt:
  expr SEMI                        { Expr($1) }
| IF LPAREN expr RPAREN stmt_list %prec NOELSE ENDIF        { If($3, List.rev $5, [Block([])],
[Block([])] ) }
| IF LPAREN expr RPAREN stmt_list ELSE stmt_list ENDIF          { If($3, List.rev $5, [Block([])],
List.rev $7) }
| IF LPAREN expr RPAREN stmt_list elseif_list ENDELSEIF ELSE stmt_list ENDIF  { If($3,
List.rev $5, List.rev $6, List.rev $9) }
| WHILE LPAREN expr RPAREN DO stmt_list ENDWHILE            { While($3, List.rev $6) }
| RETURN expr SEMI { Return($2) }


expr:
  expr PLUS   expr      { Binop($1, Add, $3) }
| expr MINUS  expr      { Binop($1, Sub, $3) }
| expr TIMES  expr      { Binop($1, Mul, $3) }
| expr DIVIDE expr      { Binop($1, Div, $3) }
| expr EQ           expr           { Binop($1, Equal, $3) }
| expr NEQ    expr      { Binop($1, Neq, $3) }
| expr LESS   expr      { Binop($1, Less, $3) }
| expr LEQ    expr      { Binop($1, Leq, $3) }
| expr GREATER expr { Binop($1, Greater, $3) }
| expr GEQ       expr   { Binop($1, Geq, $3) }
| expr CONCAT expr  { Binop($1, Concat, $3) }
| MINUS expr %prec UMINUS { Neg($2) }
| NOT expr      { Not($2) }
| LINSERT expr RINSERT { Insert($2) }
| LDELETE expr RDELETE { Delete($2) }
| LQUERY expr  RQUERY  { Query($2) }
| ID ASSIGN expr       { Assign($1, $3) }
| ID LBRACKET expr RBRACKET ASSIGN expr    { ListAssign($1, $3, $6) }
| ID LBRACKET expr RBRACKET { Mem($1, $3) }
```

```
| LBRACKET list RBRACKET { List(List.rev $2) }
| LPAREN expr RPAREN { AddParen($2) }
| ID LPAREN list RPAREN { Call($1, List.rev $3) }
| ID DOTOPT ID    { Property($1, $3) }
| ID DOTOPT ID ASSIGN expr   { PropertyAssign($1, $3, $5) }
| INT                          { Int($1) }
| LITERAL                      { Num($1) }
| STR          { Str($1) }
| ID                           { Id($1) }


list:
          /*nothing*/          { [] }
|         ID                           { [Id($1)] }
|         LITERAL              { [Num($1)] }
|         STR                  { [Str($1)] }
| INT      { [Int($1)] }
| list COMMA expr { $3 :: $1 }
```

src/scanner.mll

```
{ open Parser }

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf }
| '+' { PLUS }                          | '>' { GREATER }
| '-' { MINUS }                         | '<' { LESS }
| '*' { TIMES }                         | "==" { EQ }
| '/' { DIVIDE }                        | "!=" { NEQ }
| '=' { ASSIGN }                        | ">=" { GEQ }
| '"' { QUOTE }                         | "<=" { LEQ }
| '!' { NOT }                           | ','  { COMMA }
| '^' { CONCAT }                        | '.' { DOTOPT }
| '(' { LPAREN }                        | ')' { RPAREN }
| '[' { LBRACKET }                      | ']' { RBRACKET }
(*| '{' { LBRACE }                      | '}' { RBRACE }*)
| "<+" { LINSERT }                      | "+>" { RINSERT }
| "<-" { LDELETE }                      | "->" { RDELETE }
| "<?" { LQUERY }                       | "?>" { RQUERY }
| "Number" { NUMBER }                   | "String" { STRING }
| "Vertex" { VERTEX }                   | "Edge" { EDGE }
| "if" { IF }                           | "else" { ELSE }
| "elseif" { ELSEIF }                   | "endif" { ENDIF }
```

```
| "while" { WHILE }                    | "do" { DO }
| "endwhile" { ENDWHILE }              | "return" { RETURN }
| "endfunc" { ENDFUNC }                | ';' { SEMI }
(*| "in" { IN }                        | "out" { OUT }
| "in_degree" { INDEGREE }            | "out_degree" { OUTDEGREE }
| "to" { TO }                         | "from" { FROM }
| "weight" { WEIGHT }*)
| "endelseif" { ENDELSEIF }           | "List" { LIST }
| "/*" { comment lexbuf }
| ['0'-'9']+ as lit { INT(int_of_string lit) }
| ['0'-'9']+('.'['0'-'9']+)? as num { LITERAL(float_of_string num) }
| '"'[^'\n' '"']+'"' as str { STR(str) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lit { ID(lit) }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }
| eof { EOF }

and comment = parse
"*/" { token lexbuf } (* End-of-comment *)
| _ { comment lexbuf } (* Eat everything else *)
```

src/translate.ml

```
open Ast
open Ccode

module StringMap = Map.Make(String)

exception ParseError of string

let rec comparelist v1 v2 = match v1, v2 with
  [], []      -> true
| [], _
| _, []       -> false
| x::xs, y::ys -> x = y && comparelist xs ys

let standardLibrary = [ ["print_number";  "Void";   "Number"];
                        ["print_string";  "Void";   "String"];
                        ["print_vertex";  "Void";   "Vertex"];
                        ["print_edge";    "Void";   "Edge"  ];
                        ["link";          "Number"; "Vertex"; "Vertex"; "Number"];
                        ["bi_link";       "Number"; "Vertex"; "Vertex"; "Number"];
                        ["list_length";   "Number"; "List"  ];
                        ["cmp";           "Number"; "String"; "String"];
                      ]
```

```
let string_of_datatype = function
    Number -> "Number"
  | String -> "String"
  | Vertex -> "Vertex"
  | Edge   -> "Edge"
  | Void   -> "Void"
  | List   -> "List"

(* expr returns tuple (datatype, cstmt list) *)
(* e = expression
   sm = main variable map
   fm = function map
   lm = local variable map
   fname = function name *)
let rec expr (e, sm, fm, lm, fname) =
      match e with
      Str i -> ("String", [Strg i])
        | Id s ->  if(StringMap.mem s sm) then ((StringMap.find s sm), [Id s])
              else (if (StringMap.mem fname lm) then ( let lmap = (StringMap.find fname lm) in
if(StringMap.mem s lmap) then ((StringMap.find s lmap), [Id s]) else raise (ParseError (s ^ " not
declare")) )
                    else raise (ParseError (fname ^ " not declare")) )
        | Int i -> ("Number", [Int i])
        | Num n -> ("Number", [Numb n])
        | Vertex(label) -> ("Vertex", [Vertex label])
        | Edge(label)   -> ("Edge", [Edge label])
        | Binop (e1, op, e2) -> if(fst (expr(e1, sm, fm, lm, fname)) = fst (expr(e2, sm, fm, lm,
fname)) )
                    then ((fst (expr(e1, sm, fm, lm, fname))), [Binop (expr (e1, sm, fm, lm, fname) , op,
expr (e2, sm, fm, lm, fname) )])
                    else raise (ParseError "type not match")
        | Assign (id, value) -> if(fst (expr(value, sm, fm, lm, fname) ) = "" ) (*List type work around*)
                    then ( fst (expr(value, sm, fm, lm, fname) ), [Assign (id, (expr(value, sm, fm, lm,
fname) ))])
                    else(
                      if(fname = "")(*main*)
                      then(
                        if(fst (expr(value, sm, fm, lm, fname) ) = (StringMap.find id sm) )
                        then ( fst (expr(value, sm, fm, lm, fname) ), [Assign (id, (expr(value, sm, fm, lm,
fname) ))])
                        else raise (ParseError "type not match")
                      )
```

```
                    else(
                      if(fst (expr(value, sm, fm, lm, fname) ) = (StringMap.find id (StringMap.find
fname lm) ) )
                      then ( fst (expr(value, sm, fm, lm, fname) ), [Assign (id, (expr(value, sm, fm, lm,
fname) ))])
                      else raise (ParseError "type not match")
                    )
                  )
      | Keyword k -> ("Void", [Keyword k])
      | Not(n) -> (fst (expr (n, sm, fm, lm, fname)), [Not (snd (expr (n, sm, fm, lm, fname) ))])
      | Neg(n) -> (fst (expr (n, sm, fm, lm, fname)), [Neg (snd (expr (n, sm, fm, lm, fname) ))])
      | Call(s, el) ->  if(StringMap.mem s fm) then
                        (if( comparelist (List.tl (StringMap.find s fm) ) (List.map (fun x -> fst (expr(x,sm,
fm, lm, fname)) ) el ) )
                        then ( (List.hd (StringMap.find s fm) ), [Call (s, (List.concat (List.map (fun x ->
snd (expr(x,sm, fm, lm, fname)) ) el)))])
                        else raise (ParseError ("In function "^s^", arguement types do not match") ))
                       else raise (ParseError ("function " ^ s ^ " not defined") )
      | List(el) -> ("List", [List ("", (List.concat (List.map (fun x ->  snd (expr (x, sm, fm, lm, fname)
) ) el)))])
      | Mem(id, e) -> ("",[Mem (id, (snd (expr (e, sm, fm, lm, fname) ) ) )])
      | ListAssign(id, i, e) -> ("", [ListAssign (id, (snd (expr (i, sm, fm, lm, fname) ) ), (snd (expr (e,
sm, fm, lm, fname) ) ) )])
      | Insert(e) ->  if((fst (expr (e, sm, fm, lm, fname) ) ) = "String")
                  then ("Vertex", [Insert  (snd (expr (e, sm, fm, lm, fname) ) )])
                  else raise ( ParseError ("arguement of vertex insert has to be string") )
      | Query(e) ->   if((fst (expr (e, sm, fm, lm, fname) ) ) = "String")
                  then ("Vertex", [Query (snd (expr (e, sm, fm, lm, fname)))])
                  else raise ( ParseError ("arguement of vertex query has to be string") )
      | Delete(e) ->  if((fst (expr (e, sm, fm, lm, fname) ) ) = "String")
                  then ("Number", [Delete (snd (expr (e, sm, fm, lm, fname)))])
                  else raise ( ParseError ("arguement of vertex delete has to be string") )
      | Property(id, p) -> if(StringMap.mem id sm || (StringMap.mem id (StringMap.find fname
lm)) )
                        then(match p with
                          "in"   -> ("List", [Property (id, p)])
                        | "out"  -> ("List", [Property (id, p)])
                        | "in_degree"  -> ("Number", [Property(id, p)])
                        | "out_degree" -> ("Number", [Property(id, p)])
                        | "weight" -> ("Number", [Property(id, p)])
                        | "to" -> ("List", [Property(id, p)])
                        | "from" -> ("List", [Property(id, p)])
                        | "src" -> ("Vertex", [Property(id, p)])
```

```
                    | "dest" -> ("Vertex", [Property(id, p)])
                    | _ -> ("", [Property(id, p)])
                    )
                    else raise (ParseError (id ^ " not declared"))
      | PropertyAssign(id, p, e) -> if(StringMap.mem id sm || (StringMap.mem id (StringMap.find
fname lm)) )
                    then ("Void", [PropertyAssign (id, p, (snd (expr (e, sm, fm, lm, fname) ) ) )])
                    else raise (ParseError (id ^ " not declared"))
      | AddParen(e) -> ((fst (expr (e, sm, fm, lm, fname) ) ), [AddParen (snd (expr (e, sm, fm, lm,
fname) ) )])
      | Noexpr -> ("Void", [])

(* stmt returns cstmt list *)
(* st = stmt
  sm = main variable map
  fm = function map
  lm = local variable map
  fname = function name *)
let rec stmt (st, sm ,fm, lm, fname)  =
    match st with
       Block sl     -> List.concat (List.map (fun x -> stmt (x,sm, fm, lm, fname) ) sl )
      | Expr e      -> snd(expr (e, sm, fm, lm, fname)) @ [Keyword ";"]
      | If (e1, e2, e3, e4) -> (match e3 with
              [Block([])] -> (match e4 with
                 [Block([])] -> [If (snd(expr (e1, sm, fm, lm, fname) ) )] @ [Then (stmt (Block(e2),
sm, fm, lm, fname) )]
                |  _ -> [If (snd (expr (e1, sm, fm, lm, fname) ) )] @ [Then (stmt (Block(e2), sm, fm,
lm, fname) )] @ [Else (stmt (Block(e4), sm, fm, lm, fname) )]
              )
            |  (Elseif(e,s)) ::tail -> [If (snd (expr (e1, sm, fm, lm, fname) ) )] @ [Then (stmt
(Block(e2), sm, fm, lm, fname) )] @ [Elseif ( (snd (expr (e, sm, fm, lm, fname))), stmt (Block(s),
sm, fm, lm, fname) )] @ List.concat (List.map (fun x -> stmt (x,sm, fm, lm, fname) ) tail ) @ [Else
(stmt (Block(e4), sm, fm, lm, fname) )]
            |  _ -> raise (ParseError "caught parse error in if")
            )(*[Keyword "if "] @ expr e1 @ stmt e2 @  List.concat (List.map stmt e3) @ stmt e4*)
      | Elseif(e, s) -> [Elseif ( (snd(expr (e, sm, fm, lm, fname) ) ), (stmt (Block(s), sm, fm, lm,
fname) ))]
      | While(e, s) -> [While ( (snd (expr (e, sm, fm, lm, fname) ) ), (stmt (Block(s), sm, fm, lm,
fname) ) )]
      | Return e     -> [Return ( snd (expr (e, sm, fm, lm, fname) ) ) ]

(* translates Ast.program to cstmt list *)
(* global = main vaiables
```

```
   statements = main statements
   functions = function declarations *)
let translate (globals, statements, functions) =

  (* translate function declaration *)
  let translatefunc (fdecl, sm, fm, lm, fname) =
    let rec arg = function
        [] -> []
      | [a] -> [Datatype a.ftype] @ [Id a.frname]
      | hd::tl -> [Datatype hd.ftype] @ [Id hd.frname] @ [Keyword ", "] @ arg tl
    in
      [Datatype fdecl.rtype] @ [Id fdecl.fname] @
      [Keyword "("] @
      arg fdecl.formals @
      [Keyword ")\r\n{\r\n"] @
      List.concat (List.map (fun x -> [Datatype x.vtype] @ snd(expr (x.value, sm, fm, lm,
fdecl.fname))) fdecl.locals ) @
      (stmt ((Block fdecl.body), sm, fm, lm, fname)) @ [Keyword "\r\n}\r\n"](*@  (* Body *)
      [Ret 0]   (* Default = return 0 *)*)

  (* translatre main statements *)
  and translatestm (stm, sm, fm, lm, fname) =
    stmt ((Block stm), sm, fm, lm, fname)

  (* translate main variables *)
  and translatestg (glob, sm, fm, lm, fname) =
    [Datatype glob.vtype] @ snd(expr (glob.value, sm, fm, lm, fname))

  (* create stringMap for main variables to their types *)
  and map varlist =
    List.fold_left
      (fun m var -> if(StringMap.mem var.vname m) then raise (ParseError ("variable " ^
var.vname ^ " already declared in main"))
      else StringMap.add var.vname (string_of_datatype var.vtype) m) StringMap.empty varlist

  (* create StringMap for functions to their return types and arguement types *)
  and functionmap fdecls =
    let fmap = List.fold_left
      (fun m fdecl -> if(StringMap.mem fdecl.fname m) then raise (ParseError ("function " ^
fdecl.fname ^ " already declared") )
      else StringMap.add fdecl.fname ([string_of_datatype fdecl.rtype] @ (List.map (fun x ->
(string_of_datatype x.ftype) ) fdecl.formals) ) m) StringMap.empty fdecls
    in  List.fold_left (fun m x -> StringMap.add (List.hd x) (List.tl x) m) fmap standardLibrary
```

```
  (* create StringMap for functions to map its local variables and types *)
  and localmap fdecls =
    (* add formals first and the local variables *)
    let perfunction formals = List.fold_left
        (fun m formal -> if(StringMap.mem formal.frname m) then raise (ParseError ("formal
arguement " ^ formal.frname ^ " already declared") )
        else StringMap.add formal.frname (string_of_datatype formal.ftype) m) StringMap.empty
formals
    in
    List.fold_left
      (fun m fdecl -> if(StringMap.mem fdecl.fname m) then raise (ParseError ("function " ^
fdecl.fname ^ " already declared") )
      else StringMap.add fdecl.fname

      (List.fold_left (fun m local -> if(StringMap.mem local.vname m) then raise (ParseError ("local
variable " ^ local.vname ^" already declared in " ^ fdecl.fname) )
      else StringMap.add local.vname (string_of_datatype local.vtype) m) (perfunction
fdecl.formals)  fdecl.locals)

      m) StringMap.empty fdecls


  in [Keyword "#include \"prtzl.h\"\r\nstruct graph* g;\r\n"] @
     List.concat (List.map (fun x -> translatefunc (x, StringMap.empty, (functionmap functions),
(localmap functions), x.fname ) ) (List.rev functions) ) @
     [Main] @
     List.concat (List.map (fun x -> translatestg (x, (map globals), (functionmap functions),
StringMap.empty, "" ) ) (List.rev globals) ) @
     translatestm ((List.rev statements), (map globals), (functionmap functions), StringMap.empty,
"" )
     @ [Endmain]

(* convert cstmt to c code *)
let rec string_of_ccode (ty, cs) =
  match cs with
  Main -> "int main() {\r\ng=init_graph();\r\n"
  | Endmain -> "\r\n\t return 0; \r\n}\r\n"
  | Strg(l) -> l
  | Id(s) -> s
  | Numb(n) -> (string_of_float n)
  | Int(i) -> (string_of_int i)
  | Vertex(l) -> l
```

```
  | Edge(l) -> l
  | Keyword(k) -> k
  | Datatype(t) -> (match t with
          Number  -> "double "
        |  String  -> "char* "
        |  Vertex  -> "struct node* "
        |  Edge  -> "struct node* "
        |  List  -> "struct list* "
        |  Void  -> "void ")
  | Binop((ty1,e1),op,(ty2,e2)) -> (match op with
           Add   ->  if(ty1 = "Number" && ty2 = "Number") then
                      (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty1, x) ) e1)
)
                      ^ " + " ^
                      (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty2, x) ) e2)
)
                    else raise (ParseError "+ operator only applies to Number" )
        |  Sub   ->  if(ty1 = "Number" && ty2 = "Number") then
                      (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty1, x) ) e1)
)
                      ^ " - " ^
                      (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty2, x) ) e2)
)
                    else raise (ParseError "- operator only applies to Number" )
        |  Mul   ->  if(ty1 = "Number" && ty2 = "Number") then
                      (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty1, x) ) e1)
)
                      ^ " * " ^
                      (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty2, x) ) e2)
)
                    else raise (ParseError "* operator only applies to Number" )
        |  Div   ->  if(ty1 = "Number" && ty2 = "Number") then
                      (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty1, x) ) e1)
)
                      ^ " / "  ^
                      (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty2, x) ) e2)
)
                    else raise (ParseError "/ operator only applies to Number" )
        |  Equal ->  if(ty1 = "Number" && ty2 = "Number") then
                      (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty1, x) ) e1)
)
                      ^ " == " ^
```

```
                (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty2, x) ) e2)
)
              else raise (ParseError "== operator only applies to Number" )
      |  Neq   ->  if(ty1 = "Number" && ty2 = "Number") then
                (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty1, x) ) e1)
)
                ^ " != " ^
                (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty2, x) ) e2)
)
              else raise (ParseError "!= operator only applies to Number" )
      |  Less  ->  if(ty1 = "Number" && ty2 = "Number") then
                (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty1, x) ) e1)
)
                ^ " < " ^
                (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty2, x) ) e2)
)
              else raise (ParseError "< operator only applies to Number" )
      |  Leq   ->  if(ty1 = "Number" && ty2 = "Number") then
                (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty1, x) ) e1)
)
                ^ " <= " ^
                (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty2, x) ) e2)
)
              else raise (ParseError "<= operator only applies to Number" )
      |  Greater ->  if(ty1 = "Number" && ty2 = "Number") then
                  (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty1, x) )
e1) )
                  ^ " > " ^
                  (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty2, x) )
e2) )
                else raise (ParseError "> operator only applies to Number" )
      |  Geq   ->  if(ty1 = "Number" && ty2 = "Number") then
                (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty1, x) ) e1)
)
                ^ " >= "  ^
                (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty2, x) ) e2)
)
              else raise (ParseError ">= operator only applies to Number" )
      |  Concat  ->  if(ty1 = "String" && ty2 = "String") then
                  "cat(" ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode
(ty1, x) ) e1) )
                  ^ ", " ^
```

```
                    (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty2, x) )
e2) ) ^ ")"
                  else raise (ParseError "^ operator only applies to constant strings" )
      )
  | Assign(id, (ty,value))  -> (match value with
        []      -> id ^ ";"
      | [Strg _]  -> id ^ " = " ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode
(ty, x) ) value) ) ^ ";"
      | [Id _]    -> id ^ " = " ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode
(ty, x) ) value) ) ^ ";"
      | [Int _]   -> id ^ " = " ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode
(ty, x) ) value) ) ^ ";"
      | [Numb _]  -> id ^ " = " ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x ->
string_of_ccode (ty, x) ) value) ) ^ ";"
      |   [Vertex _]  -> id ^ ";"
      |   [Edge _]  -> id ^ ";"
      | [Keyword _] -> id ^ " = " ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x ->
string_of_ccode (ty, x) ) value) ) ^ ";"
      | [Not _]   -> id ^ " = " ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode
(ty, x) ) value) ) ^ ";"
      | [Neg _]   -> id ^ " = " ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode
(ty, x) ) value) ) ^ ";"
      | [Call _]  -> id ^ " = " ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode
(ty, x) ) value) ) ^ ";"
      |   [List (i, e)] -> id ^ " = list_init();\r\n"^ (List.fold_left (fun x y -> x^"list_add("^id^",
"^y^");\r\n") "" (List.map (fun x -> string_of_ccode (ty, x) ) e))
      |   [Binop _]   -> id ^ " = " ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x ->
string_of_ccode (ty, x) ) value) ) ^ ";"
      |   [Mem _]   -> id ^ " = " ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x ->
string_of_ccode (ty, x) ) value) ) ^ ";"
      |   [Insert _]  -> id ^ " = " ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x ->
string_of_ccode (ty, x) ) value) ) ^ ";"
      |   [Query _]   -> id ^ " = " ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x ->
string_of_ccode (ty, x) ) value) ) ^ ";"
      |   [Delete _]  -> id ^ " = " ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x ->
string_of_ccode (ty, x) ) value) ) ^ ";"
      | [Property _]  -> id ^ " = " ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x ->
string_of_ccode (ty, x) ) value) ) ^ ";"
      |   _ -> raise (ParseError "caught parse error in Assign")
      )
  | Not(cl)   -> "!(" ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty, x) )
cl)) ^ ")"
```

```
  | Neg(cl)   -> "-(" ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty, x) ) cl)) ^ ")"
  | Call(s, cl) -> s ^ "(" ^ (List.fold_left (fun x y -> match x with "" -> y | _ -> x^","^y) "" (List.map (fun x -> string_of_ccode (ty, x ) ) cl)) ^ ")"
  | List(id, cl)-> (List.fold_left (fun x y -> match x with "" -> y | _ -> x^","^y) "" (List.map (fun x -> string_of_ccode (ty, x) ) cl))
  | Mem(id, e)  -> "list_get(" ^ id ^ ", " ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty, x) ) e)) ^ ")"
  | ListAssign(id, i, e) -> "list_set(" ^ id ^ ", " ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty, x) ) e)) ^ ", " ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty, x) ) i)) ^ ")"
  | If(s)     -> "if(" ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty, x) ) s)) ^ ")"
  | Then(s)   -> "{\r\n\t" ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty, x) ) s)) ^ "\r\n}"
  | Else(s)   -> "else\r\n{\r\n\t" ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty, x) ) s)) ^ "\r\n}"
  | Elseif(e, s)-> "else if(" ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty, x) ) e)) ^ ")\r\n{\r\n\t" ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty, x) ) s)) ^ "\r\n}"
  | While(e, s) -> "while(" ^(List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty, x) ) e)) ^ ")" ^
            "{\r\n\t" ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty, x) ) s)) ^ "\r\n}"
  | Return(s)   -> "return " ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty, x) ) s)) ^ ";"
  | Insert(e)   -> "insert_vertex(g, " ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty, x) ) e)) ^ ")"
  | Query(e)  -> "query_vertex(g, " ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty, x) ) e)) ^ ")"
  | Delete(e)   -> "delete_vertex(g, " ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty, x) ) e)) ^ ")"
  | Property(id, p) -> "get_node_property("^ id ^", \"" ^ p ^ "\")"
  | PropertyAssign(id, p, e) -> "put_node_property(" ^ id ^ ", \"" ^ p ^ "\", " ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty, x) ) e)) ^ ")"
  | AddParen(e) -> "(" ^ (List.fold_left (fun x y -> x^y) "" (List.map (fun x -> string_of_ccode (ty, x) ) e)) ^ ")"


  (*| Formal(f) -> (List.fold_left (fun x y -> x^y) "" (List.map string_of_ccode f))*)
(*let _ =
  let lexbuf = Lexing.from_channel stdin in
  let program = Parser.program Scanner.token lexbuf in
```

```
  (*Translate.translate program*)
  let result = translate program in
    (List.iter (fun x -> print_endline (string_of_ccode x)) result)*)
```

src/clibs/list.c

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

#include "list.h"



struct list* list_init(){

	struct list* ret = (struct list*) malloc(sizeof(struct list));
	ret->size = 0;
	ret->head = NULL;
	ret->tail = NULL;

	return ret;
}

void list_add(struct list* l, void* data){

	struct list_entry* new_ent =
		(struct list_entry*) malloc(sizeof(struct list_entry));

	new_ent->data = data;
	new_ent->next = NULL;

	// empty list
	if(l->size == 0){
		l->head = new_ent;
		l->tail = new_ent;
	}

	// list has things
	else{
		l->tail->next = new_ent;
```

```
		l->tail = new_ent;
	}

	(l->size)++;

}

int check_boundaries(struct list* l, int i){
	return i >= 0 && i < l->size;
}

void* list_get(struct list* l, double index){

	int i = (int) index;

	if(!check_boundaries(l, i)){
		return NULL;
	}

	struct list_entry* ptr = l->head;

	void* ret = NULL;

	int j;
	for(j=0; j<i; j++){
		ptr = ptr->next;
	}

	return ptr->data;

}

int list_set(struct list* l, void* data, double index){

	int i = (int) index;

	if(!check_boundaries(l, i)){
		return 0;
	}

	struct list_entry* ptr = l->head;
```

```
	int j;
	for(j=0; j<i; j++){
		ptr = ptr->next;
	}

	ptr->data = data;

	return 1;

}

int list_remove(struct list* l, double index){

	int i = (int) index;

	if(!check_boundaries(l, i)){
		return 0;
	}

	struct list_entry* ptr = l->head, *prev;

	int j;
	for(j=0; j<i; j++){
		prev = ptr;
		ptr = ptr->next;
	}

	if(l->size == 1){

		l->head = NULL;
		l->tail = NULL;
		free(ptr);
	}

	else if(ptr == l->head){

		l->head = ptr->next;

		free(ptr);
	}

	else if(ptr == l->tail){
```

```
                l->tail = prev;
                prev->next = NULL;

                free(ptr);
        }

        else{

                prev->next = ptr->next;

                free(ptr);
        }

        (l->size)--;

        return 1;
}

int list_length(struct list* l){

        return l->size;
}

void list_destroy(struct list* l){

        while(l->size > 0){
                list_remove(l, 0);
        }

        free(l);
}

// int main(){

//      struct list* mylist = list_init();

//      int thing = 8;
//      list_add(mylist, &thing);

//      int thing2 = 9;
//      list_add(mylist, &thing2);

//      int thinggetted = *((int*) list_get(mylist, 0));
```

```
//      printf("%d %d\n", thinggetted, list_length(mylist));

//      int thing3 = 42;
//      list_add(mylist, &thing3);

//      printf("%d\n", list_length(mylist));

//      list_remove(mylist, 1);

//      int thinggetted2 = *((int*) list_get(mylist, 1));

//      printf("%d %d\n", thinggetted2, list_length(mylist));

//      list_destroy(mylist);

//      printf("list destroyed\n");

//      return 0;
// }
```

src/clibs/list.h

```
// linked list data structure
struct list{

        int size;
        struct list_entry* head;
        struct list_entry* tail;
};

// entry of the linked list
struct list_entry{

        void* data;
        struct list_entry* next;
};

struct list* list_init();

// add to end of list
void list_add(struct list*, void*);
```

```
// get specified index
// boundaries checked
// returns NULL if out of bounds
void* list_get(struct list*, double);

// sets specified index
// boundaries checked
// returns 1 if successful
int list_set(struct list*, void*, double);

// attempt to remove index
// return 1 if successful
int list_remove(struct list*, double);

// get the list length
int list_length(struct list*);

// destroy the list
void list_destroy(struct list*);
```

src/clibs/map.c

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

#include "map.h"

int hash(char* key){
	return (strlen(key) * 17) % MAP_BUCKETS;
}

struct map* map_init(){

	struct map* ret = (struct map*) malloc(sizeof(struct map));

	ret->size = 0;

	ret->buckets = (struct map_entry**) malloc(sizeof(struct map_entry**) *
MAP_BUCKETS);

	int i;
	for(i=0; i<MAP_BUCKETS; i++){
```

```
		ret->buckets[i] = NULL;
	}

	return ret;

}

void map_put(struct map* m, char* key, void* value){

	//remove existing key first
	map_del(m, key);

	int hashd = hash(key);

	struct map_entry* new_entry = (struct map_entry*) malloc(sizeof(struct map_entry));

	new_entry->key = key;
	new_entry->value = value;
	new_entry->next = NULL;

	struct map_entry* ptr = m->buckets[hashd];

	//empty bucket
	if(ptr == NULL){

		m->buckets[hashd] = new_entry;

	}

	else{

		while(ptr->next != NULL){
			ptr = ptr-> next;
		}

		ptr->next = new_entry;

	}

	(m->size)++;

}
```

```
void* map_get(struct map* m, char* key){

	int hashd = hash(key);

	struct map_entry* ptr = m->buckets[hashd];

	struct map_entry* ret = NULL;

	// key is not in the map, as key's bucket is empty
	if(ptr == NULL){

		return NULL;
	}
	//iterate bucket, try to find key
	else{

		do{

			// keys are same
			if(strcmp(key, ptr->key) == 0){
				ret = ptr;
				break;
			}

			ptr = ptr->next;

		} while(ptr != NULL);
	}

	return (ret == NULL) ? NULL : ret->value;

}

int map_del(struct map* m, char* key){

	int ret = 0;

	int hashd = hash(key);

	struct map_entry* ptr = m->buckets[hashd], *last = NULL;

	//iterate bucket, try to find key
	if(ptr != NULL){
```

```
                do{

                        // keys are same
                        if(strcmp(key, ptr->key) == 0){

                                //perform the break;
                                if(last != NULL){
                                        last->next = ptr->next;
                                }
                                //this is list head, need to update map struct
                                else{
                                        m->buckets[hashd] = ptr->next;
                                }

                                free(ptr);

                                m-> size--;

                                ret = 1;

                                break;
                        }

                        last = ptr;
                        ptr = ptr->next;

                } while(ptr != NULL);
        }

        return ret;

}

void map_destroy(struct map* m){

        int i;
        for(i=0; i<MAP_BUCKETS; i++){

                struct map_entry* ptr = m->buckets[i], *tmp;

                if(ptr != NULL){
```

```
                        do{
                                tmp = ptr->next;
                                free(ptr);
                                ptr = tmp;

                        } while(ptr->next != NULL);
                }
        }

        free(m);
}


// int main(){

//      struct map* mymap = map_init();

//      int thing = 7;
//      map_get(mymap, "candy");
//      map_put(mymap, "candy", &thing);
//      map_put(mymap, "a", &thing);
//      map_put(mymap, "b", &thing);
//      map_put(mymap, "c", &thing);
//      printf("size %d\n", mymap->size);

//      int* myval = (int*) map_get(mymap, "candy");
//      printf("get %d\n", *myval);

//      int other = 8;
//      map_put(mymap, "candy", &other);
//      printf("size %d\n", mymap->size);
//      myval = (int*) map_get(mymap, "candy");
//      printf("get %d\n", *myval);


//      int del = map_del(mymap, "candy");
//      int del2 = map_del(mymap, "candy");
//      printf("size %d %d %d\n", mymap->size, del, del2);
//      printf("contains candy? %d\n", map_get(mymap, "candy") != NULL);

//      map_destroy(mymap);

//      printf("destroyed\n");
```

```
//          return 0;
// }
```

src/clibs/map.h

```
#define MAP_BUCKETS 8

// hash map data structure
struct map{
        int size;
        struct map_entry** buckets;
};

// contains a key value entry in the map
struct map_entry{
        char* key;
        void* value;
        struct map_entry* next;
};

// initializes an empty map
struct map* map_init();

// add a key to the map, if the key exists, the current value
// is overwritten
void map_put(struct map*, char*, void* );

// gets a key from the map, if the key does not exists returns NULL
void* map_get(struct map*, char*);

// delete a key from the map, returns 1 if successful
int map_del(struct map*, char*);

// destroys the map
// using the map after destroy is undefined behavior
void map_destroy(struct map*);
```

src/clibs/prtzl.c

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

```
#include <string.h>

#include "prtzl.h"


struct graph* init_graph(){

	struct graph* ret = (struct graph*) malloc(sizeof(struct graph));
	ret->vertices = map_init();

	return ret;

}

struct node* _init_vertex(char* label){

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

	ret->properties = map_init();
	put_node_property(ret, "label", label);

	struct list* in = list_init();
	struct list* out = list_init();

	double* in_degree = (double*) malloc(sizeof(double));
	*in_degree = 0;

	double* out_degree = (double*) malloc(sizeof(double));
	*out_degree = 0;

	put_node_property(ret, "in", in);
	put_node_property(ret, "out", out);
	put_node_property(ret, "in_degree", in_degree);
	put_node_property(ret, "out_degree", out_degree);

	return ret;
}

struct node* insert_vertex(struct graph* g, char* label){

	struct node* new_vert = _init_vertex(label);

	map_put(g->vertices, label, new_vert);
```

```
	return new_vert;

}

struct node* query_vertex(struct graph* g, char* label){

	return map_get(g->vertices, label);

}

struct node* _destroy_vertex(){
	//TODO fix big memory leaks from node property map
}

double delete_vertex(struct graph* g, char* label){

	return (double) map_del(g->vertices, label);

}

void put_node_property(struct node* v, char* key, void* val){

	map_put(v->properties, key, val);

}

void* get_node_property(struct node* v, char* key){

	return map_get(v->properties, key);

}

struct node* _init_edge(struct node* src, struct node* dest, double weight){

	struct node* ret = (struct node*) malloc(sizeof(struct node));
	ret->properties = map_init();

	put_node_property(ret, "src", src);
	put_node_property(ret, "dest", dest);

	double* w = (double*) malloc(sizeof(double));
	*w = weight;
```

```
	put_node_property(ret, "weight", w);

	return ret;

}

double link(struct node* src, struct node* dest, double weight){

	struct node* new_edge = _init_edge(src, dest, weight);

	struct list* src_out = get_node_property(src, "out");
	list_add(src_out, new_edge);

	double* src_out_deg = get_node_property(src, "out_degree");
	(*src_out_deg)++;

	struct list* dest_in = get_node_property(dest, "in");
	list_add(dest_in, new_edge);

	double* dest_in_deg = get_node_property(dest, "in_degree");
	(*dest_in_deg)++;

	return 1;

}

double bi_link(struct node* src, struct node* dest, double weight){

	link(src, dest, weight);
	link(dest, src, weight);

	return 1;

}

void print_number(double n){

	printf("%lf\n", n);
}

void print_string(char* s){

	printf("%s\n", s);
```

```
}

void print_vertex(struct node* v){

	printf("%s: %.0lf edge(s) in %.0lf edge(s) out\n",
		(char*) get_node_property(v, "label"),
		*((double*) get_node_property(v, "in_degree")),
		*((double*) get_node_property(v, "out_degree")));

}

void print_edge(struct node* e){

	printf("src: %s dest: %s weight: %lf\n",
		(char*) get_node_property(get_node_property(e, "src"), "label"),
		(char*) get_node_property(get_node_property(e, "dest"), "label"),
		*((double*) get_node_property(e, "weight")));

}

char* cat(char* a, char* b){

	int len = strlen(a) + strlen(b) + 2;
	char* ret = (char*) malloc(sizeof(char) * len);
	sprintf(ret,"%s%s", a, b);

	return ret;
}

double cmp(char* a, char* b){

	if(a == NULL || b == NULL){
		return 0;
	}

	int res = strcmp(a, b);

	if(res == 0){
		return 1;
	}
	else{
		return 0;
	}
```

```
}
```

src/clibs/prtzl.h

```
#include "map.h"
#include "list.h"

// graph data type
struct graph{
	struct map* vertices;
};

// used to represent vertices and edges
// vertices and nodes have different built in properties
// that differentiate them
struct node{
	struct map* properties;
};

struct graph* init_graph();

/* <+ "key" +> */
// add a vertex to the graph with the given label
struct node* insert_vertex(struct graph*, char*);

/* <? "key" ?> */
// check if a vertex specified by the given label exists in the graph
// if not, returns NULL
struct node* query_vertex(struct graph*, char*);

/* <_ "key" _> */
// deletes a vertex from the graph
double delete_vertex(struct graph*, char*);

// attach a property to the node's property map
void put_node_property(struct node*, char*, void*);

// read a property from the node's property map
// if the property doesn't exist, returns NULL
void* get_node_property(struct node*, char*);

// links 2 vertices from first to second node with given weight
```

```
double link(struct node*, struct node*, double weight);

// creates a bidirectional link between 2 vertices with given weight
double bi_link(struct node*, struct node*, double weight);

// prints the given number to stdout
void print_number(double);

// prints the given string to stdout
void print_string(char*);

// prints a summary of the given vertex to stdout
void print_vertex(struct node*);

// prints a summary of the given edge to stdout
void print_edge(struct node*);

// concatenate 2 new strings in a new buffer
char* cat(char*, char*);

// compare 2 strings, returns 1 if they match
double cmp(char*, char*);
```

demo/demo.prtzl

```
Number dfs(Vertex v)

  List out_list;
  Number i;
  Number len;
  String visited;
  Edge e;
  Vertex dest;

  v.visited = "true";
  print_vertex(v);
  out_list = v.out;
  len = list_length(out_list);
  i = 0;

  while( i < len) do

    e = out_list[i];
```

```
    dest = e.dest;
    visited = dest.visited;

    if( !cmp("true", visited) )
      dfs(dest);
    endif

    i = i + 1;

  endwhile

  return 0;

endfunc

Vertex a = <+ "a" +>;
Vertex b = <+ "b" +>;
Vertex c = <+ "c" +>;
Vertex d = <+ "d" +>;
Vertex e = <+ "e" +>;
Vertex f = <+ "f" +>;
Vertex h = <+ "h" +>;

link(a, b, 1);
link(a, c, 1);
link(b, d, 1);
link(b, e, 1);
link(c, f, 1);
link(c, h, 1);

dfs(a);
```

.travis.yml

```
language: c

notifications:
  email:
    - m.mallett@columbia.edu
    - gl2483@columbia.edu
    - doctor.rusty.nelson@gmail.com

install:
  - sudo add-apt-repository ppa:avsm/ocaml41+opam11 -y
```

```
  - sudo apt-get update -q
  - sudo apt-get install ocaml ocaml-native-compilers camlp4-extra opam
  - sudo apt-get install rubygems
  - npm install
  - gem install rake

script:
  - rake all
```

Rakefile

```
#this test suite needs ruby to run
# \curl -sSL https://get.rvm.io | bash

task :compile do

	puts 'Compiling Ocaml source'

	Dir.chdir 'src'
	`make`
	`cp compile ../test/compile`
	Dir.chdir '..'

	puts
end

task :test do

	puts 'Running tests'
	puts

	#fresh copy over c libraries
	`cp src/clibs/* test/`

	Dir.chdir 'test'

	Dir.glob('*.prtzl').each do |file|

		name = file.chomp '.prtzl'

		puts name

		print "   Compiling .prtzl"
```

```
                if system "./compile < #{name}.prtzl > #{name}_test.c"
                        puts '    OK'
                else
                        puts '    FAILED'
                end

                print "   Compiling .c"
                if system "gcc -o #{name}.o #{name}_test.c prtzl.c map.c list.c"
                        puts '              OK'
                else
                        puts '              FAILED'
                end

                print "   Running .o"
                if system "./#{name}.o > #{name}_test.out"
                        puts '              OK'
                else
                        puts '              FAILED'
                end

                system "diff #{name}.out #{name}_test.out"
        end

        Dir.chdir '..'

        puts

end


task :clean do
        Dir.chdir 'src'
        `make clean`
        Dir.chdir '..'
        `rm test/compile test/*.o test/*_test.out test/*_test.c`
end

task :all => [:compile, :test, :clean] do

end
```

README.md

```
# prtzl
```

[![Build Status](https://img.shields.io/travis/doctorrustynelson/prtzl/master.svg)](http://travis-ci.org/doctorrustynelson/prtzl)

Overview
graph programming language compiler

test/arithmetic_addition.out
3.170000

test/arithmetic_addition.prtzl
```
Number v = 2.17 + 1;
print_number(v);
```

test/arithmetic_division.out
3.000000

test/arithmetic_division.prtzl
```
Number v = 6 / 2;
print_number(v);
```

test/arithmetic_multiplication.out
4.340000

test/arithmetic_multiplication.prtzl
```
Number v = 2.17 * 2;
print_number(v);
```

test/arithmetic_pemdas_1.out
29.000000

test/arithmetic_pemdas_1.prtzl
```
Number v = 3 + 5 * 5 + 1;
print_number(v);
```

test/arithmetic_pemdas_2.out
-3.000000

test/arithmetic_pemdas_2.prtzl
```
Number v = 3 - 25 / 5 - 1;
print_number(v);
```

test/arithmetic_pemdas_3.out

11.000000

test/arithmetic_pemdas_3.prtzl
```
Number v = ( 25 - 3 ) / ( 3 - 1 );
print_number(v);
```

test/arithmetic_subtraction.out
1.170000

test/arithmetic_subtraction.prtzl
```
Number v = 2.17 - 1;
print_number(v);
```

test/comments.out

test/comments.prtzl
```
/*

comments do nothing
adfjkadflk;j a
adfjkasdjfklads
adsflka


*/
```

test/elseif_1.out
Equals 1

test/elseif_1.prtzl
```
Number one = 1;

if(one == 1)
	print_string( "Equals 1" );
elseif(one = 2)
	print_string( "Equals 2 );
else
	print_string( "Not Equals 1 or 2" );
endif
```

test/elseif_2.out
Equals 2

test/elseif_2.prtzl

```
Number one = 2;

if(one == 1)
	print_string( "Equals 1" );
elseif(one = 2)
	print_string( "Equals 2 );
else
	print_string( "Not Equals 1 or 2" );
endif
```

test/elseif_3.out

```
Not Equals 1 or 2
```

test/elseif_3.prtzl

```
Number one = 3;

if(one == 1)
	print_string( "Equals 1" );
elseif(one = 2)
	print_string( "Equals 2 );
else
	print_string( "Not Equals 1 or 2" );
endif
```

test/empty.out

test/empty.prtzl

test/functions.out

test/functions.prtzl

```
Number one()
	return 1;
endfunc

Number add_one(Number x)
	return x + one();
endfunc

Number z = 3.1;
```

```
Number y = add_one(z);

Number w = add_one(3.1415);

String useful_function(String a, String b)
	return a ^ b;
endfunc

String hello = "hello ";
String world = "world";

String hello_world = useful_function(hello, world);
```

test/graph_ops.out

test/graph_ops.prtzl

```
String houston = "houston";
String dallas = "dallas";

Vertex v = <+ houston +>;
Vertex c = <+ "austin" +>;
<? dallas ?>;
<- houston ->;
```

test/hello.out

test/hello.prtzl

```
"hello world!";
Number myfunc() "in my function."; return 0; endfunc
```

test/if.out

test/if.prtzl

```
Number one = 1;

String x;

if(one == 1)
	x = "a";
elseif(one >= 2)
```

```
        x = "b";
elseif(one < 3)
        x = "c";
endelseif
else
        x = "d";
endif
```

test/if_else_1.out

```
Equals 1
```

test/if_else_1.prtzl

```
Number one = 1;

if(one == 1)
        print_string( "Equals 1" );
else
        print_string( "Not Equals 1" );
endif
```

test/if_else_2.out

```
Not Equals 1
```

test/if_else_2.prtzl

```
Number one = 3;

if(one == 1)
        print_string( "Equals 1" );
else
        print_string( "Not Equals 1" );
endif
```

test/if_simple_1.out

```
Equals 1
```

test/if_simple_1.prtzl

```
Number one = 1;

if(one == 1)
        print_string( "Equals 1" );
endif
```

test/if_simple_2.out

Not Equals 1

test/if_simple_2.prtzl

```
Number one = 3;

if(one != 1)
        print_string( "Not Equals 1" );
endif
```

test/link.out

```
a: 0 edge(s) in 2 edge(s) out
b: 2 edge(s) in 1 edge(s) out
c: 2 edge(s) in 1 edge(s) out
```

test/link.prtzl

```


Vertex a = <+ "a" +>;
Vertex b = <+ "b" +>;
Vertex c = <+ "c" +>;

link(a, b, 1);
link(a, c, 2);

bi_link(b, c, 3);

print_vertex(a);
print_vertex(b);
print_vertex(c);
```

test/list.out

test/list.prtzl

```


List my_list = [];

my_list[0] = "hello";
my_list[1] = "world";

String hello = my_list[0];
String world = my_list[1];
```

test/numbers.out

test/numbers.prtzl
```
Number one = 1 + 3.14;
Number two = one - 4.216 * 14;
Number three = one / two;
```

test/print.out
```
3.141593
Hello, World!
a: 0 edge(s) in 0 edge(s) out
```

test/print.prtzl

```
print_number(3.1415926);
print_string("Hello, World!");

Vertex v = <+ "a" +>;

print_vertex(v);
```

test/print_number.out
```
2.170000
```

test/print_number.prtzl
```
print_number(2.17);
```

test/print_string.out
```
hello world!
```

test/print_string.prtzl
```
print_string("hello world!");
```

test/print_vertex.out
```
vertex: 0 edge(s) in 0 edge(s) out
```

test/print_vertex.prtzl
```
Vertex v = <+ "vertex" +>;
print_vertex(v);
```

test/properties.out
```
value
```

test/properties.prtzl

```
Vertex v = <+ "v" +>;

v.key = "value";

String s = v.key;

print_string(s);
```

test/strings.out

test/strings.prtzl

```
String a = "Hello!";
String b = ", World!";
String c = a ^ b ^ "It's a beautiful day";
```

test/while.out

test/while.prtzl

```
Number i = 0;

List my_list = [];

while (i < 5) do

        my_list[i] = "loop";
        i = i + 1;

endwhile


i = 4;
String s;

while (i >= 0) do

        s = my_list[i];
        print_string(s);
        i = i - 1;
```

```
endwhile
```