YAGL: Yet Another Graph Language
Proposal

Anthony Alvarez (aea2161), David Ding (dwd2112)
Columbia University

July 11, 2016

Contents

1 Introduction

We seek to make a language, YAGL (Yet Another Graph Language), which allows users to interact with graphs in a manner similar to the language of mathematical proofs. This should allow a person with a theoretical understanding of graphs to engage with them in an intuitive practical way.

With YAGL, users will be able to easily create graphs and add vertices and edges, and associate arbitrary attributes with them (for example, colors with vertices and weights with edges). Common graph algorithms will be able to be written cleanly and concisely, as if they came out of the classic Algorithms textbook by CLRS.

2 Language Design and Syntax

This section provides a rough idea of the design of YAGL, and is subject to change. Final language design and syntax will be provided in the language reference manual.

2.1 Comments

YAGL will use Python-style comments for single line, and will not have a special syntax for multi-line comments. Example:

```
# This line is a comment.
```

2.2 Data Types

Basic types:

Type	Explanation
int	integer
float	floating point number
string	sequence of characters

YAGL also implements a notion of infinity and negative inifinty using the keywords INF and -INF respectively. INF is greater than every int or float and is equal to INF. -INF is less than every int or float and is equal to -INF.

YAGL implements constants True and False which evaluate to 1 and 0 respectively. These constants add syntactic sugar to algorithms though they are directly replaceable with ints.

Additionally we implement a Null which can take the place of int, float, or string. Null comparisons are always false. To detect nulls YAGL implements an isNull built in function.

Collection types:

Type	Explanation
list	a list of values, all of the same type
map	a map of from keys (alphanumeric strings) to values of arbitrary types
set	a set of values, all of the same type

Graph types:

Type	Explanation
Graph	an undirected graph
Digraph	a directed graph

2.2.1 Elaboration on Graphs

A graph has vertices and edges. Fundamentally, the edges in a graph are either all undirected or all directed, so we have the graph types Graph and Digraph. Vertices will be accessed by labels, and edges will be accessed via two vertex labels.

One will be able to associate any vertex or any edge with any number of attributes. These attributes will be keyed by alphanumeric, and have a value of arbitrary type. In particular, note that edge weights are not a fundamental attribute within the language, but one will be able to easily define an edge weight via, say, an attribute with name 'w' and either a float or int value.
Vertices will have the immutable attribute 'label', and edges will have the immutable attributes 'orig' and 'dest', representing the origin and destination vertices. These attributes will be defined when the vertex or edge is created.

2.3 Operators

2.3.1 Basic Operators

Category	Data Type	Operator	Explanation
Comparison	int, float, string	==, !=, >, <, <=, >=	Act the same as C++ operators.
Computation	int, float	+,-,*,/,%	Act the same as C++ operators.
Computation & Assignment	int, float	+=,-=,*=,/=	Act the same as C++ operators.
Concatenation	string	+	Concatenates two strings.
Concatenation & Assignment	string	+=	Concatenates right hand side string to original and assigns.
Boolean	int, float	!	0 maps to 1, and all other values map to 0.
Boolean	int, float	AND	1 if both values are nonzero, and 0 otherwise.
Boolean	int, float	OR	0 if both values are zero, and 1 otherwise.

2.3.2 Collection Operators

Category	Data Type	Operator	Explanation
Comparison	list, map, set	==, !=	If all values in all indices/keys are equal then == returns True else != returns True.
Contains	list, map, set	in	Returns if an item exists in the list,map, or set
Concatenation	list, set	+	Concatenates two lists or sets. On sets removes duplicates.
Concatenation, Assignment	list, set	+=	Concatenates right hand side list to original and assigns
Removal	set, map	-	Removes all items in the right hand set, or map (by key), from the left hand set if they exist
Removal, Assignment	set, map	-=	Removes all items from in the right hand set, or map (by key), from the left hand set if they exist and assigns back to left hand side set.
Access	list	[i]	Access the ith element of a list (zero-indexed)
Access	map	myMap.literal or myMap[“literal’] or myMap[variable]	YAGL supports multiple map access methods for ease of use. myMap.literal is identically equivalent to myMap[“literal’]. The bracket notation must be used if accessing using a a variable.

2.3.3 Graph Operators

Category	Data Type	Operator	Explanation
Comparison	Graph, Digraph	==, !=	If all attributes in all vertices and edges are equal then == returns True else != returns True.
Concatenation	Graph, Digraph	+	Concatenates two graphs together into a single graph. Note, names of vertices in both graphs must be distinct to concatenate them.
Concatenation & Assignment	Graph, Digraph	+=	Concatenates two graphs together into a single graph. Note, names of vertices in both graphs must be distinct to concatenate them.

2.4 Built-in Functions

Code	Explanation
print(arg1,arg2,...)	print out a comma separated list of variables and a new line character
size(iterable)	return the size of iterable
isEmpty(iterable)	return size(iterable) > 0
isNull(arg)	returns 1 if the arg is Null 0 otherwise
enqueue(list, elem)	add an element to the end of a list
dequeue(list)	remove the first element of a list and return it
push(list, elem)	add an element to the end of a list
pop(list)	remove the last element of a list and return it
adj(graph, v)	return the set of vertices in a graph adjacent to v

2.5 Control Flow

Code	Explanation
if(condition) # If condition code else # Else condition code	if-else block
while(condition) # While condition code	while loop
forEach(item, iterable) # code operating on each item of iterable	simple for loop
forComponentValue(key, value, iterable) # code operating on each key-value pair	enumerating for loop

2.6 Function Definition

Functions will be defined using the keyword def and the syntax below and the return keyword will return any output of the function

```
def myFunction( Arg1, Arg2 )
    # The code of my function
    return( 'This is the return value of myFunction()' )
```

3 Sample Code

```
## Simple collection manipulation

# Define a List
a = []
push( a, 1 )
push( a, 2 )
a == [1,2]
>> 1
pop( a )
>> 2
a
>> [1]
enqueue( a, 3 )
a
>> [1,3]
dequeue(a)
>> 1

# Define a map
c = {||}
c.key1 = 'Value1'
c.key2 = [ 0, 1, 3 ]
c.key3 = 10
print( size( c ) )
>> 3

# Define a set
d = {}
forEach( elem, c.key2 )
    d += elem
print( d )
>> {0,1,3}
d += 3
print( d )
>> {0,1,3}
d -= { 0, 1, 3, 4 }
print( isEmpty( d ) )
>> 1

## Build and examine a basic Graph
G = Graph()
G.V += 'a'
G.V += {'b','c'}
forEach( v, G.V )
    G.E += [v,v]
    G.E[v,v].weight = 1
G.E += [ 'b','c' ]
print( G.E[ 'b', 'c' ].weight )
```

```
>> Null

#Create a copy of G as a digraph
D = Digraph()
forComponentValue( label, attributes, G.V )
    D.V += label
    forComponentValue( key, value, attributes )
        D.V[key] = value
forEach( edge, G.E )
    D.E += [ edge.orig.label, edge.dest.label ]
    D.E += [ edge.dest.label, edge.orig.label ]
    forComponentValue( key, value, edge )
        D.E[ edge.orig.label, edge.dest.label ][ key ] = value
        D.E[ edge.dest.label, edge.orig.label ][ key ] = value

###############################################
# BFS takes as arguments a graph or digraph G #
# and s is a string representing the label of #
# the desired root vertex                     #
###############################################
def BFS( G, s )
    #Takes a graph G and a label for a start node s
    forEach( v, G.V )
        if( v.label == s )
            v.color = 'gray'
            v.d = 0
            v.parent = NULL
        else
            v.color = 'white'
            v.d = INF
            v.parent = NULL
    queue = []
    enqueue( queue, G.V[s] )
    while( !isEmpty( queue ) )
        u = dequeue( queue )
        forEach( v, adj( G, u ) )
            if v.color == 'white'
                v.color = 'gray'
                v.d = u.d + 1
                v.parent = u
                enqueue( queue, u )
        u.color = 'black'
    return( G )

###############################################
# Relax is a helper function for BellmanFord  #
# it takes as arguments a graph or digraph G  #
# and an edge e in that graph                 #
###############################################
def Relax( G, e )
```

```
    #If the distance of is more than the weight of the edge u->v and the distance on u
    v = e.dest
    u = e.orig
    if( v.d > u.d + e.weight )
        v.d = u.d + e.weight
        v.parent = u
    return()

###############################################
# The BellmanFord algorithm takes a graph or  #
# a digraph G and a label for a source s and  #
# returns True if there is a negative weight  #
# cycle that is reachable from s              #
#                                             #
# returns False if there is not a negative    #
# weight cycle that is reachable from s       #
###############################################
def BellmanFord( G, s )
    ret = True
    forEach( v, G.V )
        v.d = INF
        v.parent = NULL
    G.V[s].d = 0
    forEach( i, range( 0, size( G.V ) ) )
        forEach( edge, G.E )
            Relax( G, edge )
    forEach( edge, G.E )
        if( edge.dest.d > edge.orig.d + edge.w )
            ret = False
    return( ret )

```

YAGL Quick Start Guide

Welcome to YAGL! Here is some essential information to get you started:

Types:

- Numbers: any number, integer or floating point; there is no distinction. Additionally, INF is greater than any other number, and -INF is less than any other number.
    - Initialization: 1, -2.3, INF, etc.
- Strings: any string of alphanumeric characters and spaces, between single quote marks.
    - Initialization: ‘hello world’, ‘12345’, etc.
- Lists: a list containing elements of arbitrary type.
    - Initialization: [elem1, elem2, ...]
- Maps: a map from string keys to values of arbitrary type.
    - Initialization: {| ‘key1’ := 42, ‘key2’ = ‘foo’, ‘key3’ = [1, 2, 3] |}
- Graphs: a directed graph containing vertices and edges, which can be associated with arbitrary attributes.
    - Initialization: the keyword ‘graph’. That’s it!

Operators:

- = to assign variables
- +, -, *, /, % for number arithmetic
- + for string concatenation
- ==, != for number and string equality testing
- >, >=, <, <= for number comparison
- and, or, not for zero vs. nonzero numbers. The keyword ‘true’ is equivalent to the number 1 and the keyword ‘false’ is equivalent to the number 0.
- If lst is a list, the ith element (zero-indexed) can be accessed using lst[i].
- If m is a map, the key k can be accessed using m[k]. Additionally, if a map m has an attribute keyed by string ‘s’, it can be accessed using either m.s or m[‘s’].

Comments: comments fall between the /* and */ characters.

Statements:

- Any statement is terminated by a newline in YAGL.
- If/else statements work as follows. The else statement is optional:

```
if( test ) {
  ...
}
else {
  ...
}
```

- While statements work as follows:

```
while( test ) {
  ...
}
```

- You can iterate through the indices of a list (in order) or the keys of a map (in no guaranteed order) as follows:

```
forKey( i, lst ) {
  ...
}
```

or

```
forKey( k, m ) {
  ...
}
```

- You can iterate through the index/value pairs of a list (in order) or the key/value pairs of a map (in no guaranteed order) as follows:

```
forKeyValue(i, val, lst) {
  ...
}
```

or

```
forKeyValue(k, v, m) {
  ...
}
```

- To return from a function, use return (to return nothing) or return(expr) to return expr.

Functions:

- Functions are defined as follows:

```
def func(param1, param2, ...) {
  ...
}
```

- Every program must have a function called ‘main’ which takes no arguments and returns nothing.
- Functions are called using func(param1, param2, ...).
- Several functions are built in:
    - print(x), where x is a num or a string
    - remove(k, n) where k/n is an integer/list or a string/map pair. Removes k from n and returns it.
    - add(k, n) where k/n is an integer/list or a string/map pair. Adds k to n.
    - typeOf(x) which returns the type of x. This should be avoided if possible.
    - Graph functions:
        - addV(G, v), where G is a graph and v is a string. Adds a vertex with label v to G.
        - addE(G, a, b), where G is a graph and v is a string. Adds a directed edge from a to b.
        - v(G) returns a map from vertex labels to vertex attributes.
        - e(G) returns a map from edge labels (vertex labels concatenated together with a ‘~’ character in between) to edge attributes.

As an example, here is a program that defines a graph and runs Dijkstra’s algorithm:

```
/* Given a list of vertices, return the index of the vertex with the lowest
  value in the 'dist' attribute */
def minDistU( vertices ){
  minVertex = 0
  qCounter = 0
  minDistSoFar = INF
  forKeyValue( i, v, vertices ){
    thisDist = v.dist
    if( thisDist < minDistSoFar ) {
      minDistSoFar = thisDist
      minVertex = qCounter
    }
    qCounter = qCounter + 1
  }
  return( minVertex )
}

/* Run Dijkstra's algorithm from the source vertex to every other vertex */
def dijkstras( G, source ){
 Q = []

 forKeyValue( label, v, v( G ) ){
   v.dist = INF
```

```
      v.prev = NULL
      append( v(G)[ label ], Q )
    }

    v(G)[source].dist = 0

    while( not isEmpty( Q ) ){
      u = remove( minDistU(Q), Q )

      forKeyValue( i, v, adj( G, u ) ) {
        alt = u.dist + e(G)[ edgeLabel( u, v ) ].length
        if ( alt < v.dist ) {
          v.dist = alt
          v.prev = u
        }
      }
    }
  }

  def main() {
    G = graph

    /* Construct the graph */
    e1 = addE( G, 'a', 'b' )
    e2 = addE( G, 'a', 'c' )
    e3 = addE( G, 'a', 'f' )
    e4 = addE( G, 'b', 'c' )
    e5 = addE( G, 'b', 'd' )
    e6 = addE( G, 'c', 'f' )
    e7 = addE( G, 'c', 'd' )
    e8 = addE( G, 'd', 'e' )
    e9 = addE( G, 'f', 'e' )
    e1.length = 7
    e2.length = 9
    e3.length = 14
    e4.length = 10
    e5.length = 15
    e6.length = 2
    e7.length = 11
    e8.length = 6
    e9.length = 9

    dijkstras( G, 'a' )
    print( 'Shortest distances' )
    print( 'a to a' )
    print( v( G ).a.dist )
    print( 'a to b' )
    print( v( G ).b.dist )
    print( 'a to c' )
    print( v( G ).c.dist )
    print( 'a to d' )
    print( v( G ).d.dist )
    print( 'a to e' )
    print( v( G ).e.dist )
    print( 'a to f' )
    print( v( G ).f.dist )

    /* Get the shortest path from vertex a to vertex e */
    u = v( G ).e
    path = [ 'e' ]
    while ( not isEqual( u.prev, NULL ) ) {
      u = v( G )[ u.prev.label ]
      push( u.label, path )
    }
    print( 'Shortest path from vertex a to vertex e' )
    while( size( path ) > 0 ) {
      print( pop( path ) )
    }
  }
```

YAGL: Yet Another Graph Language
Language Reference Manual

Anthony Alvarez (aea2161), David Ding (dwd2112)
Columbia University

July 20, 2016

Contents

1 Preface

This is the language reference manual of YAGL (Yet Another Graph Language). It is heavily adapted from the GNU C reference manual.

2 Lexical Elements

2.1 Identifiers

Identifiers are sequences of characters used for naming variables and functions. They are alphanumeric and case-sensitive, and the first character cannot be a digit.

2.2 Keywords

The following identifiers are reserved for use as keywords, and may not be used for any other purpose:

```
and, or,
if, else,
def, return, print,
true, false, INF, NULL,
forKey, forKeyValue, while,
graph
```

2.3 Constants

There are two types of constants: number and string constants.

2.3.1 Number constants

A number constant is a sequence of digits, optionally followed by a decimal point, optionally followed by more digits, and optionally preceded by a `-` character to negate it. Note, that number constants are modeled as double precision floats.

The keywords `true` and `false` are equal to the numbers `1` and `0` respectively. Additionally, the keyword `INF` is greater than every number and equal to itself. Analogously, `-INF` is less than every integer and equal to itself.

2.3.2 String constants

A string is a sequence of characters within two `'` characters. Only, alphanumeric characters can be used in a string. Note, YAGL does not support escape sequences.

2.4 Operators

An operator is a special token that performs an operation on some number of operands. Full coverage of these can be found in a later section.

2.5 Comments

The characters /* introduce a comment, which terminates with the characters */. Multiline use the /* */ syntax for comments. YAGL also allows for single line comments using the characters `//` to begin a comment and a newline `\n` to end a comment.

2.6 Separators

A separator separates tokens. White space is a separator, but not a token. The other separators are:

```
( ) [ ] {| |} , . :=
```

2.7 White Space and New Lines

White space refers to the tab, carriage return, and space characters. They are ignored (outside of string constants) except for the purpose of separating tokens. No white space is required between operators and operands, or around separators. Separating other tokens requires any nonzero amount of white space.

New lines indicate the end of a statement. There is no way to break a long statement onto multiple lines. Programers should be careful to avoid ending statements with a comment as this will not allow the statement to terminate with a newline character and will result in invalid programs.

3 Data Types

YAGL supports two primative types: Numbers and Strings, two container types: Lists and Maps and a graph types. Note functions are not objects in YAGL, they cannot be members of containers or passed as variables to other functions.

3.1 Dynamic Typing

YAGL is dynamically typed: the type of any variable is not explicitly declared, but rather is inferred at runtime.

3.2 Primitive Types

YAGL has two primitive types:

1. Number. YAGL has just one number type, representing real numbers. There is no distinction between integers and floating-point numbers. YAGL has three number aliases, `INF` which is greater than all numbers, `true` which is 1 and `false` which is 0.

   Numbers are declared like

   ```
   3
   2.
   1.543
   INF
   true
   ```

2. String. A string consists of zero or more characters enclosed in single quotes. Only alphanumeric characters and ~ are valid within a string.

   Strings are declared like

   ```
   ''
   'string'
   'v1~v2'
   ```

3.3 The NULL type

The keyword NULL may be assigned to any variable.

3.4 Container Types

YAGL has two container types:

1. Lists. This is simply an ordered list of values of any type (not necessarily the same). Lists are declared like

   ```
   []
   [1,2,'three']
   ```

2. Maps. This is a set of key-value pairs, where a value is accessed through the key. Keys are YAGL stirngs (alphanumeric with the addition of ~) without spaces and are unique at a given level (nested keys can be repeated); values can have arbitrary types.

   Maps are declared as follows:

   ```
   {||}
   {| 'key1' := 'value1', 'key2' := 2, 'key3' := {| 'nestedKey' := [] |} |}
   ```

In general these methods of constructing objects can be applied recursively.

3.5 Graph Type

This is the standard mathematical notion of a *directed* graph: a set of vertices with string labels, and a set of directed edges each going between two vertices. Vertices must have unique labels within a graph, and there can only be one edge from one specific vertex to another specific vertex. It is possible to attach arbitrary values of any type to any vertex or edge, keyed by a string.
Graphs are declared using the keyword `graph`:

```
G = graph
```

Information on how to add vertices and edges to graphs and access them will follow later, in the built-in functions section.

4 Expressions and Operators

Precedence and associativity are discussed in a section entitled Operator Precedence.

4.1 Expressions

An expression consists of at least one operand and zero or more operators. Operands are typed objects such as constants, variables, and function calls that return values. Here are some examples:

```
42
3 + 5
-1
not true
```

Parentheses group sub expressions:

```
( 2 * ( ( 3 + 4 ) - ( 3 * 7 ) ) )
```

Innermost expressions are evaluated first. In the above example, 3 + 4 and 3 * 7 evaluate to 7 and 21, respectively. Then 21 is subtracted from 7, resulting in -14. Finally, -14 is multiplied by 2, resulting in -28. The outermost parentheses are completely optional.
An operator specifies an operation to be performed on its operand(s). Operators may have one or two operands, depending on the operator.

4.2 Assignment Operators

YAGL provides only one assignment operator, `=`. The assignment operator stores a copy of value on the right hand side into variable on the left hand side. YAGL does not support assignment to Num, String, List, Set, Map, Graph or Digraph.
Additionally, assignment expressions evaluate to the right hand side value to allow for multiple assignments. Here are some examples

```
/* Literal Assignment */
a = 10
b = 'foobar'
d = c = 12.3 /* d and c now equal 12.3 */

/* Collection Assignment */
x = [1,0,3]
z = {| 'key1' := 'value1' |}

/* Graph Assignment */
G = graph
```

4.3 Arithmetic Operators

YAGL provides operators for standard arithmetic operations: addition, subtraction, multiplication, and division, along with modular division and negation. Usage of these operators is straightforward; here are some examples:

```
/* the expression evaluates to the value in the comments to the right */
/* Addition */
x = 5 + 3  /* 8. */
y = 10.23 + 37.332 /* 47.562 */
z = x + y /* 55.562 */

/* Subtraction */
```

```
x = 5 - 3 /* 2.0 */
y = 57.223 - 10.903 /* 46.32 */
z = x - y /* -44.32 */

/* Multiplication */
x = 5 * 3 /* 15. */
y = 47.4 * 1.001 /* 47.4474 */
z = x * y /* 711.711 */

/* Division. */
x = 5 / 3 /* 1.666666667 */
y = 940.0 / 20.2 /* 46.534653465 */
z = x / y /* 0.035815603 */
```

YAGL uses the modulus operator % to obtain the remainder produced by dividing its two operands. You put the operands on either side of the operator, and it does matter which operand goes on which side: `3 % 5` yields the remainder of 3 divided by 5 which is 3, while `5 % 3` yields the remainder of 5 divided by 3 which is 2. The operands must be expressions resolving to numbers.

```
/* the expression evaluates to the value in the comments to the right */
/* Modular division */
w = 5 % 3 /* 2. */
x = 74.23 % 47 /* 27.23 */
y = -74.23 % 47 /* 19.77 */
z = 2 % 1.1 /* 0.9 */
```

YAGL provides a negation operator for mapping a number to its negative counterpart.

```
/* Negation */
x = -5 /* -5 */
y = -( 4 * 0.4 ) /* -1.6 */
```

4.4 Equality Testing Operators

YAGL uses the equality testing operators `==` and `!=` to determine if two items are equivalent. The result of these operators is either 1 or 0, meaning true or false, respectively.

Note equality operators only tests equality of primitive operands (number, string, and NULL) to test the equality of collections or graphs one should use the standard library function isEqual(a,b) which will be discussed in a later section.

4.4.1 Equality and Non Equality

The equal-to operator `==` tests its two operands for equality. The result is false if the operands are of different types or are collections or graphs. If the operands are primatives of the same type `==` returns true if they have the same value, and false otherwise.

```
x = 10
y = 10
x == y /* True */
x = {| 'key1' := 10, 'key2' := 20 |}
```

```
    y = {| 'key1' := 10, 'key2' := 20 |}
    x == y /* False because == does not validate equality on collections */
```

The not-equal-to operator `!=` tests its two operands for inequality. The result is always the opposite of the result for `==`

```
    x = 20
    y = 30
    x != y /* true */
    x = [ 1, 2, 3, 4 ]
    y = [ 1, 2, 3, 4 ]
    x !=y /* True because == does not validate equality on collections and != always returns the opp
```

4.4.2 Other Comparisons

Beyond equality and inequality, YAGL supports, less than, greater than, less-than-or-equal-to, or greater-than-or-equal-to another operators on numbers using the operators `<,>,<=,>=` respectively. Here are some code samples that exemplify usage of these operators.

```
    x = 10
    y = 15

    x < y /* True */
    10 * x <= y /* False */

    -x > -y /* True */
    x >= y - 5 /* True */
```

4.5 Logical Operators

Logical operators test the truth value of of numerical expression operands. YAGL considers any expression which evaluates to a non zero number to be true while an expression that evaluates to zero YAGL considers false. The result of any logical operators is either 1 or 0, meaning true or false, respectively.
The logical conjunction operator `and` tests if two expressions are both true. It returns 1 (true) when both expressions are true and 0 otherwise. Note, YAGL does not perform any logical short circuiting, both expressions are always evaluated.

```
    x = 5
    y = 10
    x == 5 and y == 10 /* True */
    x == 5 and 0 /* False */
```

The logical conjunction operator `or` tests if at least one of two expressions it true. It returns 1 (True) when either expressions are true and 0 otherwise. Note, YAGL does not perform any logical short circuiting, both expressions are always evaluated.

```
    x = 5
    y = 10
    x == 15 or y == 10 /* True */
    x == 10 or y == 5 /* False */
```

You can prepend any numerical expression with a negation operator `not` to flip the truth value.

```
x = 0
not x == 0 /* False */
not x /* True */
```

4.6 String Operators

In addition to the comparison operators, YAGL also support string concatenation using the + operator.

```
x = 'foo'
y = 'bar'
z = x + y
z = z + 'baz'
/* z now equals 'foobarbaz' */
```

4.7 List Operators

List access is done through a `list-expr[int-expr]` where `list-expr` is an expression which evaluates to a list and `int-expr` evaluates to an integer number. While YAGL does not have integers explicitly this is checked at run time, so users should take care to pass in integer values.

```
x = [1,2,[3]]
y = x[0] /* y is 1 */
z = x[2][0] /* z is 3 */
```

4.8 Map Operators

In addition to comparison operators, YAGL has two syntaxes for accessing and assigning keys and values of maps. When combined with the assignment operator the access method can be used to add new keys and values to maps or update the value of an already existing key.

The first is a uses brackets like `map-expression[string-expression]` where `map-expression` is an expression which evaluates to a map and `string-expression` is any expression which evaluates to a string. This notation will return the value in the `map-expression` associated with they key `string-expression`. Below is an example.

```
a = {| 'key1' := 1 |}
a['key2'] = 2
x = 'key3'
a[x] = 3
/* a now equals the map {| 'key1' := 1, 'key2' := 2, 'key3' := 3 |} */
```

The second syntax adds a bit of sugar to the first and uses a period delimiter after a map variable name like `var.literal` which is identically equivalent to `var['literal']`. Note, that this syntax does not evaluate arbitrary expressions but rather simply converts everything from the period to the next space into a literal string. Below is an example

```
a = {| 'key1' := 1 |}
a.key2 = 2
/* a now equals the map {| 'key1' := 1, 'key2' := 2 |} */
```

4.9 Graph and Digraph Operators

There are no special operators on graphs and digraphs, there are several built in function calls related to graphs which will be discussed in a later section.

4.10 Operator Precedence

Operator precedence works as follows, from highest to lowest. Operators with equal precedence get applied left-to-right.

1. list/map item access
2. `not`, negation
3. *, /,
4. +, -
5. <, <=, >, >=
6. ==, ! =
7. `and`
8. `or`
9. Assignment

5 Statements

Statements cause actions and control flow within programs. Statements can be written to do nothing at all, or something completely trivial.

5.1 Expression Statements

Any expression becomes a statement when it is followed by a new line character. It is possible for an expression to have no effect; for example:

```
5
2 + 2
10 >= 9
```

These are useless because they do nothing except for the evaluation itself, and do not store the value anywhere. Some examples of more useful statements:

```
x = 25
x = x + 25
y = [ 1, 2, 3 ]
```

5.2 The if statement

The if statement can be used to conditionally execute part of the program, based on the truth value of a given expression. Here is the general form of an if statement:

```
if( test ) {
  then-statements
}
else {
  else-statements
}
```

The `test` is an expression evaluating to a number, any nonzero number indicates true and zero indicates false. The `then-statements` and `else-statements` blocks are any number of statements, including zero.
If `test`, evaluates to true (any nonzero number), then the statements in the `then-statements` block are executed and `else-statements` block is not. If `test` evaluates to false (zero), then `else-statements` block is executed and `then-statements` block is not. The else clause is optional.

While the indentation is not required, it is good style to indent then and else statements
Here is an actual example:

```
if( x >= 10 ) {
  x = x + 1
}
else {
  x = x - 1
}
```

5.3 The while statement

The while statement is a loop statement with an exit test at the beginning of the loop. Here is the general form of the while statement:

```
while (test) {
  statements
}
```

The `test` is an expression evaluating to a number, any nonzero number indicates true and zero indicates false. The `statements` block is any number of statements, including zero.
The while statement first evaluates `test` which is an expression evaluating to a number. If `test` evaluates to true (any nonzero number), then `statement` is executed, and `test` is evaluated again. `statement` continues to execute repeatedly as long as `test` is true after each execution of `statement`.

The following example prints the integers from zero to nine:

```
i = 0
while( i < 10 ) {
  print( i )
  i = i + 1
}
```

5.4 The forKey statement

YAGL's `forKey` statement iterates over the keys of an iterable (list,map). For a list the keys are the indecies and for a map the keys are the keys associated with values. Items in lists are iterated in order from index 0 to the last index, However when iterating over maps/sets no guarantee is made about order of iteration.
The syntax `forKey(var,iter)` where `var` is a label for use in the loop, and `iter` is a list or map. The `forKey(var,iter)` copies the keys stored in the iterable one at a time into the variable `var`. See the below examples.

```
x = [1,2,3]
z = {| 'key1' := 'foo', 'key2' := 'bar' |}

/* List */
y = 0
forKey( item, x ){
    y = y + item
}
/* y is 3 */


/* Map */
a = ''
forKey( item, z ){
    a = a + item
}
/* a is now the string 'foobar' or the string 'barfoo'
       because YAGL makes no guarantee of iteration order over maps */
```

5.5 The forKeyValue statement

YAGL's forKeyValue statement iterates over the items stored in an iterable (list or map). Items in lists are iterated in order from index 0 to the last index. However when iterating over maps no guarantee is made about order of iteration.
Note that for a list the keys are the numerical indices. The syntax `forKeyValue(c, v, iter)` makes a shallow copy of the values stored in the iterable one at a time into the variable `v` and copies the key into the variable `c`. See the below examples.

```
/* List */
x = [10,20,30]

keySum = 0
valueSum = 0
forKeyValue(c,v,x){
    keySum = keySum + c
    valueSum = valueSum + v
}
/* keySum = 0 + 1 + 2 = 3,and valueSum = 10 + 20 + 30 = 60 */

/* Map */
z = {| 'key1' := 1, 'key2' := 2 |}
```

```
keyAccum = ''
valueAccum = 0
forKeyValue(c,v,z){
    keyAcuum = keyAccum + c
    valueAccum = valueAccum + v
}
/* keyAccum is 'key1key2' or 'key2key1' because YAGL makes no guarantee of iteration order over
   and valueAcuum = 1 + 2 = 3 */
```

If updating the container's values while itterating through a container the objective then one must be aware and careful of shallow copy nature of `forKeyValue()`. See the below examples.

```
/* Note the potential side effects of a forKeyValue due to the shallow copy into v */
x = ['one','two','three']
forKeyValue( i, var, x ) {
    var = 2
}
/* x still equals ['one','two','three'] */

x = [['one'],['two'],['three']]
forKeyValue( i, var, x ) {
    var = 2
}
/* x still equals [['one'],['two'],['three']] */

x = [['one'],['two'],['three']]
forKeyValue( i, var, x ) {
    x[i] = 2
}
/* x still equals [2,2,2] */

x = [['one'],['two'],['three']]
forKeyValue( i, var, x ) {
    var[0] = 2
}
/* x now equals [[2],[2],[2]] */
```

5.6 The return statement

The `return` statement ends the execution of a function and returns program control to the function that called it. Here is the general form of a `return` statement:

```
return( return-expr )
```

Where return-expr is any expression. It is possible to have `return` without `return-value`; in this case, execution of the function will return NULL. It is also possible to omit `return` entirely, in which case the function will run until the closing brace and return NULL.
Return statements can be placed at the end of any block of code, a block of code being defined by curly braces {}.

6 Functions

Functions separate parts of a program into distinct subprocedures.

6.1 Function Definitions

Function definitions specify what a function actually does. It consists of information regarding the function's name and parameters, along with the body of the function, which is enclosed in braces. Here is the general form of a function definition:

```
def function-name( parameter-list ) {
  function-body
}
```

Note that the opening curly brace must be on the same line as the function definition, and the closing curly brace must be on its own line.

Here is an example of a simple function, which returns the sum of the parameters:

```
def add( x, y ) {
  return( x + y )
}
```

6.2 Calling Functions

A function in YAGL is called using its name and supplying the necessary parameters, separated by commas. Here is the general form of a function call:

```
function-name( parameters )
```

A function call can make up an entire statement, or can be used as a subexpression. Here is an example of a standalone function call:

```
x = 0
f( x )
```

Standalone function calls are not necessarily useless, as parameters are always passed by reference (discussed in more detail in the next section), so the parameters that are passed in may change.

Here is an example of a function call used as a subexpression:

```
x = f( 0 )
```

If a function takes more than one argument, then parameters are separated with commas; for example:

```
x = g( 0, 1 )
```

6.3 Function Parameters

Function parameters can be any expression: a literal value, a value stored in a variable, or something else. It is important to note that parameters are passed by *reference*, not by copy. Changing a parameter within a function changes its value outside the scope of the function. For example, suppose we have the following function:

```
def f( x ) {
  x = x + 1
}
```

If `x` has value 0, and then `f(x)` is called, then `x` will have value 1 afterwards.

To avoid changing a variable when passing it into a function, one can make a copy of it first using the standard library function call `deepCopy()`.

6.4 The Main Function

Every program requires at least one function called `main`, where the program begins executing. `main` should take no parameters. `main` should not return anything; any return value is ignored.

6.5 Recursive Functions

YAGL supports recursive functions: functions which call themselves. For example, here is a function that computes the factorial of a nonnegative integer:

```
def factorial(x) {
  if (x <= 1) {
    return( 1 )
  }
  else {
    return( x * factorial(x - 1) )
  }
}
```

It is important to not write a function that is infinitely recursive, or the program will continue until it is either interrupted or runs out of memory.

6.6 Nested Functions

YAGL does not support nested functions (functions defined within other functions).

6.7 Built-in Functions

There are a few built-in functions which cannot be defined by users.

1. `print(expr)`, which can be called only on an expression evaluating to a number value (printing out the standard decimal representation) or a string value (simply printing out the string), or a null value and it will print the string `NULL`. Print also returns the value.

   ```
   print( 'Hello World' )
   print( NULL )
   x = print( 12 + 3 ) /* x now equals 15 and 15 has been printed */
   ```

2. `remove(i, l)` removes the value stored at index i from list l and returns it. `remove(k, m)` removes the value stored at key k from map m and returns it.

   ```
   a = [1,2,3]
   print( remove( 1, a ) ) /* Prints to output 2. */
   a == [1,3] /* True */
   ```

3. `append(expr,list-expr)` takes any item as a first argument and a list expression as the second adds the item to the end of the list.

4. `typeOf(expr)` takes any expression and returns a number: -1,0,1,2,3,4 indicating NULL, Number, String, List, Map or Graph respectively. This function should be avoided if possible.

Here is how to add vertices and edges to graphs and access them:

1. Adding a vertex. To add vertex 'a', use `addV(G, 'a')`. This returns a map of the vertex attributes. If a vertex is added twice, nothing happens the second time; the existing attribute map is returned.

2. Adding an edge. To add an edge from vertex 'a' to vertex 'b', use `addE(G, 'a', 'b')`. Vertices 'a' and 'b' are added automatically if they do not already exist. This returns a map of the edge attributes. If an edge is added twice, nothing happens the second time; the existing attribute map is returned.

3. Accessing a vertex. `v(G)` returns a *map* from vertices (keyed by the string used when they were added) to their corresponding attributes, which are themselves stored as a map from attribute name to value.

   Each vertex has a special attribute 'label' which returns the label of the vertex.

4. Accessing an edge. `e(G)` returns a *map* from edges (keyed by a string consisting of the edge's origin then destination vertex labels, separated by a `~`) to their corresponding attributes, which are themselves stored as a map from attribute name to value.

   Each edge has two special attributes 'orig' and 'dest' which return the attributes of the edge's origin and destination vertices respectively. Note that the vertex attributes accessed via 'v0' and 'v1' are not a copy; altering them will in fact directly alter the vertex attributes.

Here is some example code involving graph instantiation:

```
G = graph()
addV( G, 'a' )
forKey( elem, {'b','c'} ){
    addV( G, elem )
}
/* add a self edge to all vertices, and add capacity 1 to all vertices */
forKeyValue( label, attributes, G.V ) {
    addE( G, label, label )
    G.E[label + '~' + label].weight = 1
    attributes.capacity = 1
}
/* adding a new edge b~c */
addE( G, 'b', 'c' )
/* Note that the edge b~c does not have weight set. so G.E.b~c.weight == Null */
```

7 Program Structure and Scope

7.1 Program Structure

A program exists entirely within one file. All code must either be a function definition, or exist within a function block (including the requisite `main` function). Note, YAGL does not support global variables.

7.2 Scope

All functions can be called from other functions; order declaration does not matter. A variable is visible only within the function in which it is declared; once it is declared, it is visible in any function code following the declaration, including outside a loop if it was declared within a loop. For example the code

```
    def f(y) {
        x = 0.1
        print( x )
        return( y * 15 + x )
    }

    def main() {
        forKey( x, [ 100, 1000 ] ){
            print( x )
            print( f( x ) )
            print( '' )
        }
        print( x )
    }
```

produces the output

```
>>100.000000000
>>0.100000000
>>1500.100000000
>>
>>1000.000000000
>>0.100000000
>>15000.100000000
>>
>>1000.000000000
```

8 YAGL Standard library

In addition to the built ins YAGL has a standard library which encapsulates many of the features users may want in standard graph algorithms. Here is the code for those functions which is available in `std.y`.

```
// Returns true if a collection has no elements
def isEmpty( iterable ) {
  forKey( a,iterable ){
    return( false )
  }
  return( true )
}

// Returns the number of elements in a collection
def size( iterable ) {
  ret = 0
  forKey( a,iterable ){
    ret = ret + 1
  }
  return( ret )
}

// Puts an item in a Queue
def enqueue( item,list ) {
  return( append( item, list ) )
}

// Removes an item from a Queue
def dequeue( list ){
  if( isEmpty( list ) ) {
    return( NULL )
  }
  else {
    return( remove( 0, list ) )
  }
}

// Puts an item on a Stack
def push( item, list ) {
  append( item, list )
}

// Remves an item from a Stack
def pop( list ){
  if( isEmpty( list ) ) {
    return( NULL )
  }
  else {
    return( remove( size( list ) - 1, list )  )
  }
}

// Returns true if an item is Null false otherwise
def isNull( x ) {
  return( typeOf( x ) == -1 )
}

// Returns true if an item is a Number false otherwise
```

```
def isNum( x ){
  return( typeOf( x ) == 0 )
}

// Returns true if an item is a String false otherwise
def isString( x ){
  return( typeOf( x ) == 1 )
}

// Returns true if an item is a list false otherwise
def isList( x ) {
  return( typeOf( x ) == 2 )
}

// Returns true if an item is a Map false otherwise
def isMap( x ) {
  return( typeOf( x ) == 3 )
}

// Returns true if an item is a Graph false otherwise
def isGraph( x ) {
  return( typeOf( x ) == 4 )
}

// Returns true if two items are equal
// for collections returns true only if all items at all levels are identically equal
def isEqual( a, b ) {
  if( not ( typeOf( a ) == typeOf( b ) ) ) {
     return( false )
  }
  else {
    if( isNull( a )  or isNum( a ) or isString( a ) ) {
       return( a == b )
    }
    else {
      // slightly more complicated cases
      if( isList( a ) or isMap( a ) ) {
        if( size( a ) != size( b ) ) {
          return( false )
        }
        elemsEqual = 0
        forKey( elem, a ) {
          elemsEqual = elemsEqual + isEqual( a[elem], b[elem] )
        }
        // Return the equality
        return( elemsEqual == size( a ) )
      }
      else {
      // graph, only other type
        return( isEqual( v( a ), v( b ) ) and isEqual( e( a ), e( b ) ) )
      }
    }
  }
}

// Returns an item identical to a but with no references to a
def deepCopy( a ) {
  if( isNull( a ) ) {
    return( NULL )
  }
  if( isNum( a ) ) {
    return( a )
  }
  if( isString( a ) ) {
    return( a )
  }
  if( isList( a ) ) {
    b = []
    forKeyValue( i, v, a ) {
      append( deepCopy( v ), b )
    }
    return( b )
  }
  if( isMap( a ) ) {
    b = {||}
    forKeyValue( k, v, a ) {
      b[ k ] = deepCopy( v )
    }
    return( b )
  }
  if( isGraph( a ) ) {
    b = graph
    forKeyValue( label, v, v( a ) ) {
      newV = addV( b, label )
      forKeyValue( k, attr, v ) {
        newV[ k ] = deepCopy( attr )
      }
    }
    forKeyValue( label, e, e( a ) ) {
      newE = addE( a, e.orig.label, e.dest.label )
      forKeyValue( k, attr, e ) {
        newE[ k ] = deepCopy( attr )
      }
    }
    return( b )
  }
}

// Returns true if an item is stored in a list or is a key in a map
def isIn( item, iterable ){
  // iterable should be a list or a map
  if( isList( iterable ) ) {
    forKeyValue( i, val, iterable ) {
      if( isEqual( val, item ) ) {
       return( true )
      }
    }
      return( false )
  }
  if( isMap( iterable ) ) {
    forKey( k, iterable ){
      if( isEqual( k, item ) ) {
        return( true )
      }
    }
    return( false )
  }
  return( false )
}

/* Takes a graph as its first argument, and a vertex
(either its string label or its attribute map) as its
second argument, and returns the adjacent vertices in a
list of string labels if a string label was passed in
or a list of attribute maps if a map was passed in. */
def adj( G, v ) {
  res = []
  if( isString( v ) ) {
    forKeyValue(label, edge, e(G)) {
      if (edge.orig.label == v) {
        append(edge.dest.label, res)
      }
    }
  }
  if( isMap( v ) ) {
    forKeyValue(label, edge, e(G)) {
      if (edge.orig.label == v.label) {
        append(edge.dest, res)
      }
    }
  }
  return( res )
}

/*********************************************
** getEdgeLabel takes as argument two vertecies
** attribute maps and returns a string which is
** the label of the edge from the first two the
** second vertex
*********************************************/
```

```
def edgeLabel(u, v) {
  return( u.label + '~' + v.label )
}
```

9 Sample Code

```
/****************************************
** range takes as argument a number and
** returns a list from 0 to that number
**
** This function is in the YAGL standard library
****************************************/
def range( x ) {
    i = 0
    ret = []
    while( i < x ){
        enqueue( ret, i )
        i = i + 1
    }
    return( ret )
}

/*******************************************
** BFS takes as arguments a graph or digraph G
** and s is a string representing the label of
** the desired root vertex
*******************************************/
def BFS( G, s ) {
    /* Takes a graph G and a label for a start node s */
    forKey( v, v(G) ) {
        if( v(G)[v]label == s ) {
            v(G)[v]color = 'gray'
            v(G)[v]d = 0
            v(G)[v]parent = NULL
        }
        else {
            v(G)[v]color = 'white'
            v(G)[v]d = INF
            v(G)[v]parent = NULL
        }
    }
    queue = []
    enqueue( queue, G.V[s] )
    while( not isEmpty( queue ) ) {
        u = dequeue( queue )
        forKey( v, adj( G, u ) ) {
            if( v(G)[v]color == 'white' ){
                v(G)[v]color = 'gray'
                v(G)[v]d = u.d + 1
                v(G)[v]parent = u
                enqueue( queue, u )
            }
        }
        u.color = 'black'
    }
    return( G )
}

/*****************************************
** minDistU to be used in conjuction with
** Dijstras algorithm. Given a list of vertices,
** return the index of the vertex with the lowest
**   value in the 'dist' attribute
*****************************************/
def minDistU( vertices ){
   minVertex = 0
   qCounter = 0
   minDistSoFar = INF
   forKeyValue( i, v, vertices ){
      thisDist = v.dist
      if( thisDist < minDistSoFar ) {
         minDistSoFar = thisDist
         minVertex = qCounter
      }
      qCounter = qCounter + 1
   }
   return( minVertex )
}

/********************************************
** Run Dijkstra's algorithm from the source vertex
** to every other vertex
********************************************/
def dijkstras( G, source ){
  Q = []

  forKeyValue( label, v, v( G ) ){
      v.dist = INF
      v.prev = NULL
      append( v(G)[ label ], Q )
  }

  v(G)[source].dist = 0

  while( not isEmpty( Q ) ){
    u = remove( minDistU(Q), Q )

    forKeyValue( i, v, adj( G, u ) ) {
      alt = u.dist + e(G)[ edgeLabel( u, v ) ].length
      if ( alt < v.dist ) {
        v.dist = alt
        v.prev = u
      }
    }
  }
}
```

YAGL Project Plan

Process

Planning process: we started by setting some major milestones (scanner/parser done, "Hello world" program written, etc.). Within each milestone, we continuously figured out the highest priority tasks that needed to be done. As we were a team of just two people who had worked together extensively before, communication was speedy and effective.

Specification process: we began with the desired end result, inspired by graph algorithm pseudocode in the CLRS algorithms textbook. From there, we worked backwards and figured out what we needed in the language (dynamic typing, a unique map access syntax, more lenient scoping, etc.). This gave us the LRM. After we had the LRM, we largely stuck to it, modifying or removing a few features here, but overall sticking to the paradigm that we had in mind.

Development process: development began with the scanner, AST, and parser. As our language was dynamic, semantic checking was less of a priority, and took a backseat to code generation, which took up the vast majority of development time. After we had a solid set of base features implemented in code generation, we began writing the semantic checker, as well as type checking within LLVM itself.

Testing process: we adapted the testing script from the MicroC compiler and created our own suite of tests. Whenever we implemented a feature, we added a unit test, and we made sure we did not cause any major breakages before pushing code to the git repository.

Team Responsibilities

Being a group of two and working with an extremely compressed timeline, both of us had to put significant work into all aspects of the compiler. However, we did individually end up focusing more on certain aspects than others:

- Anthony Alvarez: acted more as the language guru, making final language feature decisions, and set up testing infrastructure.
- David Ding: environment setup for effective development, and got the parser/scanner and code generation off the ground into a preliminary state.

Again, despite these focuses, we would like to emphasize that we both had significant input into all aspects of the compiler, from language design to development to testing.

Project Timeline

- July 6: beginning of class
- July 11: project proposal
- July 20: LRM

- August 1: “Hello world” program
- August 11: Project presentation and final report

Software Development Environment

The development environment was largely adapted from the MicroC compiler development environment. We first got that working, and then restarted in the same environment with our own code. Versions used:

- Ocaml version 4.02.3, with
- LLVM 3.8

We coded in a few different text editors, depending on what was most convenient at the time, including nano, gedit, vim, and Sublime Text.

YAGL style Guide

Code Layout:

Functions should be defined first. The main function should be defined last. IMPORTANT: a program MUST have a blank newline at the end.

Indentation: In general consistent indentation is desirable in blocks, though not required. Nested blocks should be indented further by a consistent amount in each block. Eg.

```
def foo( bool ){
  // this is indented two spaces
  if( bool ) {
    // indent a bit further
    x = 1
  }
  else {
    x = 2
  }
  return( x )
}
```

Comments:

Comments should always be on their own line to avoid collisions with the new line characters that delimit statements. For example this is bad form

```
 def foo( bool ){
    if( bool ) { /* this is comment
                    Will cause this program to not parse
                    Because there is no new line after the { after if */
    // indent a bit further
    x = 1
  }
  else {
    x = 2
  }
  return( x )
}
```

This good form

```
 def foo( bool ){
    if( bool ) {
/* this is comment
   Will allow this program to parse
   Because there is a new line after the { after if */
    // indent a bit further
    x = 1
  }
  else {
    x = 2
  }
  return( x )
}
```

Naming Conventions:

In general YAGL functions and variables are camelCasedLikeThis.
In general variables are lowercase except when they are graphs then they are uppercase.
See the below example.

```
 def fooBar( bool, G ){
// G is a graph
    if( bool ) {
/* this is comment
   Will allow this program to parse
   Because there is a new line after the { after if */
    // indent a bit further
    x = 1
  }
  else {
    x = 2
  }
  return( x )
}
```

Blank lines:
One should always have blank lines between function declarations to increase readability and prevent issues with parsing due to missing new line characters.

Variable Names:
While one could use the character '~' in variable names in strings (particularly map keys and vertex names) best practice should avoid using this character because it is reserved for denoting the edge between two vertices.
This avoids the unnecessary confusion caused by '~' in vertex names the edge `'a~b~c'` may be ambiguous between an edge between vertex 'a' and vertex `'b~c'` or vertex `'a~b'` and vertex `'c'`.

Map Keys:
One could add keys to a graph using the bracket syntax which contained a space.
The below program is valid

```
a = {||}
a['this is my key'] = 1
```

but the key `'this is my key'` cannot be accessed using the dot method. I.e.

```
a.this is my key = 20
```

Will not parse. As such one should only use strings with no spaces to provide ease of access.

YAGL Project Log

Author	Commit	Message	Timestamp	
Anthony Alvarez	503d077	add more busness error message to scanner	36 minutes ago	
Anthony Alvarez	9bbda3e	adding commit messages in file commits... so meta	37 minutes ago	
Anthony Alvarez	b12a60a M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	54 minutes ago	
Anthony Alvarez	85ddf60	add merged test text file	55 minutes ago	
David Ding	6400882	Formatted files for sticking in a PDF	an hour ago	
Anthony Alvarez	528cc58	update test script err	an hour ago	
David Ding	51c18f7	More tests revisions; slight fix to type checking in remove.	an hour ago	
David Ding	7fb8bcd	Updated unit tests.	2 hours ago	
David Ding	92e63e8	Modifications to Dijkstra's	6 hours ago	
David Ding	63cdaf8	Dijkstra's works!	13 hours ago	
David Ding	d38a446	Updated some tests	14 hours ago	
Anthony Alvarez	91217ae	add some forKV tests	15 hours ago	
Anthony Alvarez	32d7779	removed break/continue from ast and added more type check	15 hours ago	
Anthony Alvarez	97d9368	fix small bug in while	16 hours ago	
Anthony Alvarez	520d406	add a bunch of type checking in codegen	16 hours ago	
Anthony Alvarez	6b5cd99 M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	23 hours ago	
Anthony Alvarez	fb37ca5	merge	23 hours ago	
Anthony Alvarez	5be2246	add dijk.y	23 hours ago	
David Ding	3573a9a	Renamed forEach to forKey; added forKeyValue. Updated tests.	23 hours ago	
Anthony Alvarez	0ae8889 M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	23 hours ago	
Anthony Alvarez	e644a3f	added type checking helper funcition	23 hours ago	
David Ding	9b562ea	Wrote adj	yesterday	
Anthony Alvarez	9d0dfe4	adding standard library	yesterday	
David Ding	8088be1 M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	yesterday	
David Ding	9cae4c9	Fixed issue with terminator in forEach	yesterday	
Anthony Alvarez	fe2f56f	add test for forif issue	yesterday	
Anthony Alvarez	a0038bd M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	yesterday	
David Ding	c68e945	Added label attribute to vertices.	yesterday	
Anthony Alvarez	490263a M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl Conflicts: codegen.ml	yesterday	
David Ding	2f33414	Returning at the end of forEach works	yesterday	
Anthony Alvarez	f3faa65 M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	yesterday	
Anthony Alvarez	78b7085	change to typeof and append function order	yesterday	
David Ding	634d4af	Fixed the issue where vertex/edge maps are overwritten if a vertex is readded.	yesterday	
David Ding	aaaefae	Added orig and dest attributes to edges	yesterday	
David Ding	0fb2b0c M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	yesterday	

YAGL Project Log

David Ding	ad4cdca	Improved graph capabilities.	yesterday	
Anthony Alvarez	10c075b	add ~	yesterday	
Anthony Alvarez	6c0f7d3 M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl Conflicts: semant.ml	yesterday	
David Ding	d68b8db	Updated the TODO	yesterday	
David Ding	2729505	Map remove working	yesterday	
Anthony Alvarez	64afd38	merge	yesterday	
David Ding	de91704	Revert Makefile changes	yesterday	
Anthony Alvarez	9812f40 M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	yesterday	
Anthony Alvarez	bd710b0	merge	yesterday	
Anthony Alvarez	86992a0	merge conflict	yesterday	
David Ding	f2686a8	List append works	yesterday	
David Ding	ef84417 M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	yesterday	
David Ding	6575e5f	You can now import Str in semant.ml	yesterday	
Anthony Alvarez	3de5c87	add single line comments	yesterday	
Anthony Alvarez	97270f8	update todo	2 days ago	
David Ding	c973057	Updated TODO file	2 days ago	
David Ding	bfd4cbe M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	2 days ago	
David Ding	989a6ec	FIXED THE FOR-EACH BUG!!! WOOOOO	2 days ago	
Anthony Alvarez	9b5fd27	add a thing to todo	2 days ago	
Anthony Alvarez	aa201dc	add typeof	2 days ago	
Anthony Alvarez	b83424a M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	2 days ago	
Anthony Alvarez	82303de	LIST REMOVE INDEX !~~~~ btw ! is 1 but excited	2 days ago	
David Ding	356f4b0	If an edge with uninitialized vertices is added, add the vertices automatically.	2 days ago	
David Ding	48bf936 M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	2 days ago	
David Ding	23f4437	Kill graphs in favor of digraphs (simply called graphs). Add support for adding and accessing edges.	2 days ago	
Anthony Alvarez	7c196f4	added foreach semantic checking and tests	2 days ago	
Anthony Alvarez	b52411f	add inf to semant checker	2 days ago	
David Ding	d67958b	Clean up most of the backlog of compile-time warnings.	2 days ago	
David Ding	44489bf	String concatenation works, maybe? Possibly need to set a max string length.	2 days ago	
David Ding	fd26ade	Added in a patch to list removal. It works if i > 0 and feels kinda hacky.	2 days ago	
David Ding	765613a	Added forEach for map keys, but unsurprisingly, it runs into the same issue as forEach for list indices.	2 days ago	
David Ding	13067fc	Removed a bunch of unnecessary (I think?) add_terminal blocks	2 days ago	
Anthony Alvarez	44ca45a	fixed list/map access - had some misnamed builders	2 days ago	
David Ding	c60c2f0	Fix some forEach tests (but not forEach4)	2 days ago	

YAGL Project Log

David Ding	5ab865d	Some cleanup done. Took out some unnecessary pointers and stuff.	2 days ago	
David Ding	bbc505e	Fix the formatting of a comment which is still making my text editor cry	2 days ago	
David Ding	face525 M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	2 days ago	
David Ding	affb37a	Trying to fix forEach to no avail...committing, then pulling, then cleaning up everything as it is	2 days ago	
Anthony Alvarez	3a84619	List removal doesn't work but the skeleton is there	2 days ago	
Anthony Alvarez	4e5b196	more work on remove no longer has desired side effect	2 days ago	
Anthony Alvarez	d1956be	last commit of the night	2 days ago	
Anthony Alvarez	4.42E+93	just a bit of list removal	3 days ago	
Anthony Alvarez	66c6ed2	add type checking for binop so far only num allowed	3 days ago	
Anthony Alvarez	492ea48	really fix mod and update test	3 days ago	
Anthony Alvarez	e1b067f	fixed bug in mod	3 days ago	
David Ding	fded479	Turns out alloca frees all memory that was allocated as soon as a function returns, whoops. Replace all instances of alloca with malloc. We may need to think about freeing everything at the end somehow.	3 days ago	
Anthony Alvarez	ca19c0c	comment out all of list attempt	4 days ago	
Anthony Alvarez	c9dcada M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl Conflicts: codegen.ml	4 days ago	
Anthony Alvarez	378440e	add attempt at list equality... seems to blow the stack every time	4 days ago	
David Ding	3e7cb67	forEach working; merge conflicts resolved	4 days ago	
David Ding	581eb75 M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	4 days ago	
David Ding	1082ef1	forEach loops for lists work, kinda. Need to get test-forEach4 to pass.	4 days ago	
Anthony Alvarez	070323e	added is_true helper function	4 days ago	
Anthony Alvarez	4ce4000	adding in TODO.list * items are low priority	4 days ago	
Anthony Alvarez	c57386c	STRING EQUALITY WORKS	4 days ago	
Anthony Alvarez	5f828f1	found stray basic block. now code works again	4 days ago	
Anthony Alvarez	c2e8337	committing to get a diff	4 days ago	
Anthony Alvarez	89c2b18	added a bit of abstraction to A.add laying groundwork for overload '+'	4 days ago	
Anthony Alvarez	45f8d2d M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	4 days ago	
Anthony Alvarez	f1aa08f	better type checking with more useful return for overloaded operators	4 days ago	
David Ding	2f51ac0 M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	4 days ago	
David Ding	3265f8a	Adding and accessing graph vertices works.	4 days ago	
Anthony Alvarez	500b4d9	added null support and test for null comparisions	4 days ago	
Anthony Alvarez	f83fc6c	GENERIC EQUALITY	4 days ago	

YAGL Project Log

Anthony Alvarez	528fe03 M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	4 days ago	
Anthony Alvarez	de9408c	begin making equality and neq special for comparison between types	4 days ago	
David Ding	d60007a	Maybe fix the weird segfault? If so, it was a bug in variable assignment.	4 days ago	
David Ding	5bda21c M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	4 days ago	
David Ding	9743490	Add a function to build an empty graph	4 days ago	
Anthony Alvarez	72ffdce	better inf test	4 days ago	
Anthony Alvarez	24980bd M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	4 days ago	
Anthony Alvarez	a3904fb	committing code gen for merge	4 days ago	
Anthony Alvarez	7.43E+83	add infinity	4 days ago	
David Ding	d6c3ab4	Added really basic infrastructure for graphs. Doesn't do anything significant with graphs yet.	4 days ago	
David Ding	6fcaaaf M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	4 days ago	
David Ding	6fdead2	Fix a map test.	4 days ago	
Anthony Alvarez	f582184 M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	4 days ago	
Anthony Alvarez	13962ab	print struct uses case statement. Example for ALL FUTURE CASE STATEMENTS. HUZZAH!!!	4 days ago	
David Ding	88b2dba	Fix passing by reference	4 days ago	
David Ding	2b22df8 M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	4 days ago	
David Ding	469c303	Adding keys to maps works to some degree, woo	4 days ago	
Anthony Alvarez	9bfa8ba	update test script to accomodate run time error tests add first incompatible type test	4 days ago	
Anthony Alvarez	34dd626	Ensure null comparisions are valid	4 days ago	
Anthony Alvarez	31e7cc3	type checking appears to work	4 days ago	
Anthony Alvarez	4cdfcd2 M	And fix error in type_comparision helper function Merge branch 'master' of https://bitbucket.org/comsw4115/yagl Conflicts: codegen.ml	4 days ago	
David Ding	c580d3a	Fixed most of the newly failing tests.	5 days ago	
Anthony Alvarez	08d513d M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	5 days ago	
Anthony Alvarez	508248e	attempt to make a type comparision generic throw block	5 days ago	
David Ding	475d102	Added a way to make the program crash if we want it to.	5 days ago	
David Ding	dc24ffa M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	5 days ago	
David Ding	7dc23c9	Added some convenience functions for easy type checking and retrieval.	5 days ago	
David Ding	b8bb26c	Forgot what I added here...	5 days ago	
Anthony Alvarez	040f930	add main function return regtests	5 days ago	
Anthony Alvarez	edec90e	fix return check in semantic checker	5 days ago	
Anthony Alvarez	f19f7a0	added function tests case	5 days ago	

YAGL Project Log

Anthony Alvarez	3c4f9ff	add tests for mod	5 days ago	
Anthony Alvarez	6bdd5b4 M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	5 days ago	
Anthony Alvarez	577f6d1	updating test script to accomodate OCAMLRUNPARAMS	5 days ago	
David Ding	5a22910	List and map access are equivalent; consolidate them into one parsed expression.	5 days ago	
David Ding	2444d79 M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	5 days ago	
David Ding	9e67dfe	Map access by map.string syntax works!	5 days ago	
Anthony Alvarez	f275914	add some expression tests. no output yet tbd when we do llvm level checking	5 days ago	
Anthony Alvarez	a980994	updated error message	5 days ago	
Anthony Alvarez	e01f9cc	added assignment failure tests	5 days ago	
Anthony Alvarez	7f2202b	add some fail tests	5 days ago	
Anthony Alvarez	990bb51	add expression and function semantic checking no multiple keys in maps no assignment to constants no calling functions with wrong number of arguments no returning mid block no defining print,remove or adjG	5 days ago	
David Ding	7b0d7c7	Require parentheses for return statements in parsing.	5 days ago	
David Ding	933ebc8	Assigning expr = expr now works.	5 days ago	
David Ding	98e13e4 M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	5 days ago	
David Ding	ce278f9	Miscellaneous improvements: map creation (but not access), removing the opening and closing quotes on strings	5 days ago	
Anthony Alvarez	7e18a03 M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	5 days ago	
Anthony Alvarez	c30b9e9	working basic semantic checker	5 days ago	
David Ding	0aded61	Fixed syntax errors in gcd tests	5 days ago	
David Ding	e61437a M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	5 days ago	
David Ding	a69b92c	While + tests are now working.	5 days ago	
Anthony Alvarez	9c3c0c3	reference test for maps in functions	5 days ago	
David Ding	631043	Revert "add some return (0)s": we found out LLVM only supports returning void and int types, but we'd like main to be able to return doubles, or nothing at all. This reverts commit 5c030497f6c76b1949e86e090c27b87aad06f4c6.	5 days ago	
Anthony Alvarez	5c03049	add some return (0)s	5 days ago	
Anthony Alvarez	1f86dc9	add initial map tests	5 days ago	
David Ding	4f794f5	Add a bunch of comments to codegen.	2016-08-05	
David Ding	4ca0b4d	Fix variable scoping issues by creating a block for running alloca for variables first, which then points to a block for statements.	2016-08-05	
David Ding	5c62835	If statements work for the most part. There's a bug in variable assignment that still lets the program run but makes the LLVM checker crash.	2016-08-05	
David Ding	a59d91a	All tests that should pass now pass.	2016-08-05	
David Ding	43d922c	Fixed a bug in function returns	2016-08-05	

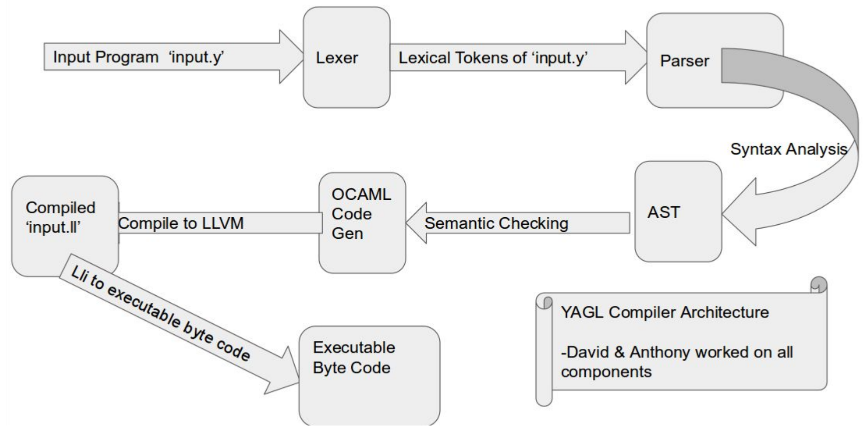
YAGL Project Log

David Ding	228899e	Fixed a bug with throwing an error in bad list indexing.	2016-08-04	
David Ding	9f7cec9	Function passing and returning implemented	2016-08-04	
David Ding	a543388	List access by expression works! Also remove a random file.	2016-08-04	
David Ding	bc409c5	Added a basic error check for bad list indices.	2016-08-04	
David Ding	99a411d M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	2016-08-03	
David Ding	13e15a8	List access by expression is partially there.	2016-08-03	
Anthony Alvarez	7179605	get some list tests	2016-08-03	
Anthony Alvarez	f9066d0	get not and negation working	2016-08-03	
Anthony Alvarez	4de3eae M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	2016-08-03	
Anthony Alvarez	5b9a193	get add and or working. Fix some test scripts	2016-08-03	
David Ding	9777e38 M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	2016-08-03	
David Ding	5c1a3c6	Bug fix - turns out you don't call free when using alloca	2016-08-03	
Anthony Alvarez	f3bb195	removing aspiration tests folder	2016-08-03	
Anthony Alvarez	f4e2fb1	update test to eliminate printb	2016-08-03	
Anthony Alvarez	5c97f6c	modified tests to work first real error caught in arith tets	2016-08-03	
Anthony Alvarez	323093d M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	2016-08-03	
Anthony Alvarez	88b50f6	updated how -d to remove default delete all output file logic	2016-08-03	
David Ding	ef1209a	Change permissions on the test script.	2016-08-03	
Anthony Alvarez	480552c	Moved a bunch of tests to aspiration_tests/* added delete argument as default to ./runtests.sh no more clutter	2016-08-03	
Anthony Alvarez	ab6c2f6	Move testing framework to runtests.sh	2016-08-03	
David Ding	2c23308	Readded ptrs that I took out a few commits ago because they're necessary to check if a field has been set. Whoops.	2016-08-03	
David Ding	7ef4f19	Added a key to the node struct, which makes it usable for maps as well.	2016-08-03	
David Ding	b57ec02	List access by nonnegative integer (secretly a float) works	2016-08-03	
David Ding	2cfcc3a	Took a pointer layer out of node structs	2016-08-03	
David Ding	36e1e43	Took a layer of pointers out of number management.	2016-08-03	
David Ding	8e99f24	Printed the first element of a list.	2016-08-03	
David Ding	05bf4ef	More arithmetic and comparisons working	2016-08-02	
David Ding	463a84d	Addition of floats works.	2016-08-02	
David Ding	8e6236e	Printing expressions of different types now works in earnest in a non-janky way. Woohoo!	2016-08-02	
David Ding	32ee7f6	Partially rewritten. Assigning to variables as well as printing floats works fine.	2016-08-01	
David Ding	2b40f25	Some more return stuff. I need to rewrite a bunch of stuff in order to let functions return arbitrary stuff.	2016-08-01	
David Ding	eefc194	main now returns void, and all other functions return an empty struct.	2016-08-01	

YAGL Project Log

David Ding	5d52250 M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	2016-08-01	
David Ding	9b5858e	Assigning and reassigning a variable to strings and numbers works, woo	2016-08-01	
Anthony Alvarez	5f9bb1c M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	2016-07-31	
Anthony Alvarez	45ec3ef	update test script to handle no returns	2016-07-31	
David Ding	9d04d0d M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	2016-07-31	
David Ding	d6b3349	Changing execution permissions on the test script	2016-07-31	
David Ding	c611431	The struct now contains a pointer to a string which can be assigned and referenced. Just need to switch between doubles and strings now...	2016-07-31	
David Ding	8da0a9b	Syntax fix in test-print3	2016-07-31	
Anthony Alvarez	f087842	updated parser to include new line characters	2016-07-31	
Anthony Alvarez	65bf81a	update floats to have .000000000	2016-07-31	
Anthony Alvarez	25ee667	commit initial tests scripts primarly driven from microC tests	2016-07-31	
Anthony Alvarez	8f74fa2	update testall.sh comments to reffer to YAGL instead of MicroC	2016-07-31	
Anthony Alvarez	f916718	adding some test scripts	2016-07-31	
David Ding	eaa62c6	Replaced the struct's double with a pointer to a double and successfully assigned and accessed it.	2016-07-31	
David Ding	42bf830	Stuck a double in a struct and successfully assigned and accessed it.	2016-07-31	
David Ding	457cca4	Variable assignment (to doubles) works.	2016-07-31	
David Ding	90c258c	Printing an arbitrary string works (printing numbers has been temporarily disabled).	2016-07-27	
David Ding	d06bce4	A PROGRAM COMPILED AND PRINTED 0.00000000!! WOOOO!!	2016-07-26	
David Ding	9fea32b	Correct a minor bug in the parser.	2016-07-25	
David Ding	83a914d M	Merge branch 'master' of https://bitbucket.org/comsw4115/yagl	2016-07-25	
David Ding	c33f9ea	Codegen now compiles; not sure it works.	2016-07-25	
Anthony Alvarez	e6214eb	Adding semant.ml directly from microc	2016-07-25	
David Ding	e36b89a	Starting to add in code generation, with the hope of getting a simple print statement to compile. Doesn't have any syntax errors; doesn't compile.	2016-07-24	
David Ding	4fe181a	Add in functions for pretty-printing a program.	2016-07-24	
David Ding	4c9232f	Added a bunch more skeleton code. It finishes if you run 'make', which is kind of nice.	2016-07-23	
David Ding	a377570	Resolved shift/reduce conflicts in the parser.	2016-07-23	
David Ding	7327955	Updated scanner, AST, parser. Still untested.	2016-07-23	
David Ding	21563a3	Partial progress on scanner, AST, parser.	2016-07-18	
David Ding	aca0b95	Empty AST and scanner files.	2016-07-18	

Block diagram of components:



5 Architectural Design

The architecture of the YAGL translator consists of the following major components: Lexer, Parser, Semantic Checker, and Code Generator, shown above.

All are implemented in OCAML, and compiled down into LLVM.
The entry point of the compiler is yagl.ml, which calls component.

First, the input YAGL code is passed to scanner.ml and parser.ml, generating tokens which then get constructed in to an AST structure which is passed
to semantic checker in semant.ml

The semantic checker ensures that the user is not doing something outrageous like assigning to a constant, failed to declare main or something of that ilk. But because YAGL is dynamically typed much of the type checking that would normally be done here is pushed down into the LLVM code itself to throw runtime errors.

Then the semantically-checked AST is passed to codegen.ml. Which then recursively defines LLVM code through the OCAML bindings leveraging heavily OCAML's ability to match on types defined by the parser and passed in through the AST. The output of the codegen.ml is LLVM code which is then compiled into byte code by lli.

Both David and Anthony worked on all components though Anthony focused more heavily on the semantic, type checking and David focused more on code generation.

6 Lessons Learned

Anthony Alvarez:

In code generation many of the helper functions we had set up initially were all build in ocaml and therefore littering the llvm with duplicate code everywhere. This made debugging the llvm nearly impossible as the programs were were compiling got a bit more complicated. This could have been solved by pushing these functionality down into LLVM functions and then calling those throughout the code. All of the llvm instructions for the helper functions would all be in one place instead of copied everywhere.

This architecture of the code generation would have also allowed us to recursively call helper functions. Particularly, there was a sneaky bug in equals where we were attempting to create an infinite amount of llvm code to check equality of lists, maps or graphs. This would have been eliminated if we could call llvm functions recursively, instead of trying to generate code recursively.

My advice to future teams is to not shy away from pushing more logic down into the llvm. It turns out it isn’t too hard and it may solve a bunch of problems later on.

David Ding:

Writing a compiler is an incredibly formidable task. As a general method of approaching a problem, it helped a lot to break it down into chunks, then break it down into more chunks until they were of a manageable size, and focus only on that task until it was done, before moving onto the next one.

Regarding implementation itself: implementing dynamic typing is not easy! Not knowing either OCaml or LLVM certainly didn’t help, but we ended up picking them up decently along the way. Ultimately, dynamic typing just pushes type checking, which is relatively easy in Ocaml, to LLVM, which makes it a lot harder. We should have invested more time into directly writing helper functions like type checkers in C, rather than try to generate code for them in Ocaml.

Test Plan

Below are all of our tests with the desired output for each.
The test suite had three primary components.

- Good programs with desired outputs denoted test-*
- Programs that would fail the semantic checker denoted fail-*
- Programs that would fail at run time due to type inconsistencies and so we know why they would fail. denoted run-*

Tests of note are dijkstras.y because it shows the language actually implementing graph algorithms.

There were some tests like dijkstras that we chose to illustrate end to end power of the language and test that all the features fit together as expected.

Other tests were unit tests of specific features, see the arithmetic or the other operation tests.

There were some tests that fell in the middle, for example the iffor tests which test behavior of if blocks inside forkey blocks. We were seeing some strange emergent behaviour there which prompted further testing.

Overall we have lots of unittests, few multi-feature tests which we created when specific issues arose and a few end-to-end tests to prove to ourselves that everything works as anticipated together and to show off the language we've built.

To ensure that running tests was easy we used a modified version of the bash script that came with microC but modified slightly to allow for runtime failures. This allowed running tests to be quick and easy so we could test them before every commit to ensure nothing was too broken.

Testing environment was mostly setup by Anthony Alvarez with strong contributions from David Ding.

```
test-while2.y
====================
def foo( a) {
  j == 0
  while (a > 0) {
    j == j + 2
    a == a - 1
  }
  return( j )
}

def main() {
  print(foo(7))
}

test-while2.out
====================
14.000000000

test-while1.y
====================
def main() {
  i == 5
  while(i > 0) {
    print(i)
    i == i - 1
  }
  print(42)
}

test-while1.out
====================
5.000000000
4.000000000
3.000000000
2.000000000
1.000000000
42.000000000

test-var1.y
====================
def main() {
  a == 42
  print(a)
}

test-var1.out
====================
42.000000000

test-remove.y
====================
def main() {
  a == [4, 5, 6]
  remove( 1, a )
  forKeyValue( k, v, a ) {
    print( v )
  }

  b == {| 'key1' :== 'hello', 'key2' :== 'world' |}
  remove( 'key2', b )
  print( b.key1 )
  print( b.key2 )
}

test-remove.out
====================
4.000000000
6.000000000
hello
NULL

test-print3.y
====================
def main() {
  print(42)
  print(71)
  print(1)
}

test-print3.out
====================
42.000000000
71.000000000
1.000000000
```

```
test-print2.y
====================
def main(){
    print( 'hello world' )
}

test-print2.out
====================
hello world

test-print1.y
====================
def main(){
    print( 1234 )
}

test-print1.out
====================
1234.000000000

test-ops4.y
====================
def main() {
  print( 'a' ==== 1 )
  print( 'a' !== 1 )
  print( 'a' ==== NULL )
  print( NULL ==== NULL )
  print( 'a' ==== 'a' )
  print( 'a' !== 'a' )
  print( 'foo' !== 'bar' )
  print( 'foo' ==== 'bar' )
}

test-ops4.out
====================
0.000000000
1.000000000
0.000000000
1.000000000
1.000000000
0.000000000
1.000000000
0.000000000
```

```
test-ops3.y
====================
def main() {
  print( 5 %2 )
  print( 72.23 % 47 )
  print( -74.23 % 47 )
  print( 2 % 1.1 )
  print( 12 % 4 )
}
```

```
test-ops3.out
====================
1.000000000
25.230000000
19.770000000
0.900000000
0.000000000
```

```
test-ops2.y
====================
def main() {
  print(1)
  print(0)
  print(true and true)
  print(true and false)
  print(false and true)
  print(false and false)
  print(true or true)
  print(true or false)
  print(false or true)
  print(false or false)
  print(not false)
  print(not true)
  print(-10)
  print(-42)

}
```

```
test-ops2.out
====================
1.000000000
0.000000000
```

```
1.000000000
0.000000000
0.000000000
0.000000000
1.000000000
1.000000000
1.000000000
0.000000000
1.000000000
0.000000000
-10.000000000
-42.000000000
```

test-ops1.y

```
====================
def main() {
  print(1 + 2)
  print(1 - 2)
  print(1 * 2)
  print(100 / 2)
  print(99)
  print(1 ==== 2)
  print(1 ==== 1)
  print(99)
  print(1 !== 2)
  print(1 !== 1)
  print(99)
  print(1 < 2)
  print(2 < 1)
  print(99)
  print(1 <== 2)
  print(1 <== 1)
  print(2 <== 1)
  print(99)
  print(1 > 2)
  print(2 > 1)
  print(99)
  print(1 >== 2)
  print(1 >== 1)
  print(2 >== 1)
}
```

test-ops1.out

```
====================
3.000000000
-1.000000000
2.000000000
50.000000000
99.000000000
0.000000000
1.000000000
99.000000000
1.000000000
0.000000000
99.000000000
1.000000000
0.000000000
99.000000000
1.000000000
1.000000000
0.000000000
99.000000000
0.000000000
1.000000000
99.000000000
0.000000000
1.000000000
1.000000000
```

test-map1.y

```
====================
def main() {
  a == {| 'key1' :== 1, 'key2' :== 'two' |}
  a.key3 == ['three']
  a.key4 == {| 'key5' :== '5' |}
  a.key4.key5 == '6'

  print( a.key1 )
  print( a.key2 )
  print( a.key3[0] )
  print( a['key4'].key5 )
}
```

test-map1.out

```
====================
1.000000000
two
three
6

test-local2.y
====================
def foo( a, b) {
  c == a
  return(c + 10)
}

def main() {
 print( foo( 37, false ) )
}

test-local2.out
====================
47.000000000

test-local1.y
====================
def foo(i) {
  /* Should reassign the formal i */
  i == 42
  print(i + i)
}

def main() {
  foo(true)
}

test-local1.out
====================
84.000000000

test-list1.y
====================
def main() {
  x == []
  y == [0,1,2]
  z == [1,'two', ['three'] ]
  print( y[0] )
  print( z[ y[0] ] )
  print( z[ y[0] + y[1] ] )
  print( z[ x == y[2] ][0] )
}

test-list1.out
====================
0.000000000
1.000000000
two
three

test-isEqual.y
====================
def main() {
  a == [ 1, 2, 'aoeu', [ 3, 4 ] ]
  b == {| 'c' :== 5, 'd' :== graph |}
  append(b, a)

  c == [ 1, 2, 'aoeu', [ 3, 4 ] ]

  print( isEqual( a, c ) )

  d == {| 'c' :== 5, 'd' :== graph |}
  append(d, c)

  print( isEqual( a, c ) )
}

test-isEqual.out
====================
0.000000000
1.000000000

test-inf1.y
====================
def main() {
  x == INF
  y == -INF
  z == 0
  print( z < x )
  print( z < y )
  print( z > x )
  print( z > y )
  print( x ==== INF )
  print( y ==== -INF )
}

test-inf1.out
====================
1.000000000
0.000000000
0.000000000
1.000000000
1.000000000
1.000000000

test-if6.y
====================
def cond(b) {
  if(b){
    x == 42
  }
  else {
    x == 17
  }
  return( x )
}

def zero(){
  return( 0 )
}

def main() {
 print(cond(10))
 print(cond(10-10))
 print(cond(zero()-1))
 print(cond(zero()))
}

test-if6.out
====================
42.000000000
17.000000000
42.000000000
17.000000000
```

```
test-if5.y
====================
def cond(b) {
  if(b){
    x == 42
  }
  else {
    x == 17
  }
  return( x )
}

def main() {
 print(cond(true))
 print(cond(false))
}

test-if5.out
====================
42.000000000
17.000000000

test-if4.y
====================
def main() {
  if(false){
    print(42)
  }
  else {
    print(8)
  }
  print(17)
}

test-if4.out
====================
8.000000000
17.000000000

test-if3.y
====================
def main() {
  if(false){
    print(42)
  }
  print(17)
}

test-if3.out
====================
17.000000000

test-if2.y
====================
def main() {
  if(true) {
    print(42)
  }
  else {
    print(8)
  }
  print(17)
}

test-if2.out
====================
42.000000000
17.000000000

test-if1.y
====================
def main() {
  if( true ){
     print(42)
  }
  print(17)
}

test-if1.out
====================
42.000000000
17.000000000

test-graph1.y
====================
def main() {
  G == graph
  addV(G,'v1')
  addV(G,'v2')
  i == 1
  forKeyValue( label, v, v(G) ){
    v.cap == i
    i == i + 1
  }
  addV(G,'v3')
  print( v(G).v1.cap )
  print( v(G).v2.cap )
  print( v(G).v3.cap )
}
```

```
test-graph1.out
====================
1.000000000
2.000000000
NULL
```

```
test-gcd2.y
====================
def gcd( a, b) {
  while (a !== b) {
    if (a > b) {
      a == a - b
    }
    else {
      b == b - a
    }
  }
  return( a )
}

def main() {
  print(gcd(14,21))
  print(gcd(8,36))
  print(gcd(99,121))
}
```

```
test-gcd2.out
====================
7.000000000
4.000000000
11.000000000

test-gcd1.y
====================
def gcd( a,  b) {
  while (a !== b) {
    if(a > b){
      a == a - b
    }
    else {
       b == b - a
    }
  }
  return( a )
}

def main() {
  print(gcd(2,14))
  print(gcd(3,15))
  print(gcd(99,121))
}

test-gcd1.out
====================
2.000000000
3.000000000
11.000000000

test-func9.y
====================
def foo(x) {
  x == 'string'
}

def main() {
   y == 10
   foo( y )
   print( y )
}

test-func9.out
```

```
====================
string

test-func8.y
====================
def foo(a) {
  print(a + 2)
}

def main() {
  foo(40)
}

test-func8.out
====================
42.000000000

test-func7.y
====================
def foo( c ) {
  a == c + 42
}

def main() {
  a == 0
  foo(100)
  print(a)
}

test-func7.out
====================
0.000000000

test-func6.y
====================
def foo(){
}

def bar(a, b, c){
  return( a + c )
}

def main() {
  print( bar(17, false, 25) )
}

test-func6.out
====================
42.000000000

test-func5.y
====================
def foo(a){
  return( a )
}

def main()
{
}

test-func4.y
====================
def add( a, b) {
  c == a + b
  return( c )
}

def main() {
  d == add(52, 10)
  print(d)
}

test-func4.out
====================
62.000000000

test-func3.y
====================
def printem(a, b, c, d){
  print(a)
  print(b)
  print(c)
  print(d)
}

def main() {
  printem(42,17,192,8)
}

test-func3.out
====================
42.000000000
17.000000000
192.000000000
8.000000000

test-func2.y
====================

def fun( x,  y) {
  return( 0 )
}

def main() {
  i == 1

  fun(i == 2, i == i+1)

  print(i)
}


test-func2.out
====================
3.000000000

test-func1.y
====================
def add( a, b) {
  return( a + b )
}

def main() {
  a == add(39, 3)
  print(a)
}

test-func1.out
====================
42.000000000

test-forKV3.y
====================
def main() {
   a == {| 'key1' :== 1,'key2' :== 2,'key3' :== 3 |}
   forKeyValue(k,v,a){
    v == k
   }
   forKeyValue(k,v,a){
    print( v )
   }
}

test-forKV3.out
====================
1.000000000
2.000000000
3.000000000

test-forKV2.y
====================
def main() {
   a == [[1],[2],[3]]
   forKeyValue(k,v,a){
    v == 'one'
   }
   forKeyValue(k,v,a){
    print( v[0] )
   }
}

test-forKV2.out
====================
1.000000000
2.000000000
3.000000000

test-forKV1.y
====================
def main() {
  a == ['foo','bar']
  forKeyValue(k,v,a){
    print( k )
    print( v )
  }
}

test-forKV1.out
====================
0.000000000
foo
1.000000000
bar

test-forKey4.y
====================
def main() {
  x == [1,2,3,4,5]
  forKey(i,x){
    x[i] == x[i] * x[i]
  }
  forKey(j,x){
    print( x[j] )
  }
  print( 42 )
}

test-forKey4.out
====================
1.000000000
4.000000000
9.000000000
16.000000000
25.000000000
42.000000000

test-forKey3.y
====================
def main() {
  x == [0,1,2,3,4]
  forKey(i,x){
  }
  print(i)
}
```

```
test-forKey3.out
====================
4.000000000

test-forKey2.y
====================
def main() {
  x == [1,2,3,4,5]
  forKey(i,x){
    print(x[i])
    i == i + 1
  }
  print(42)
}

test-forKey2.out
====================
1.000000000
2.000000000
3.000000000
4.000000000
5.000000000
42.000000000

test-forKey1.y
====================
def main() {
  x == [1,2,3,4,5]
  forKey(i,x){
    print(x[i])
  }
  print(42)
}

test-forKey1.out
====================
1.000000000
2.000000000
3.000000000
4.000000000
5.000000000
42.000000000
```

```
test-forif1.y
====================
def main() {
   a == ['1','2','3','4']
   forKey( i, a ) {
     if( i % 2 ==== 0 ){
       print( a[i] )
     }
   }
}
```

```
test-forif1.out
====================
1
3
```

```
test-fib.y
====================
def fib( x ) {
  if( x < 2 ){
     return( 1 )
  }
  return( fib( x-1 ) + fib( x-2 ) )
}

def main() {
  print(fib(0))
  print(fib(1))
  print(fib(2))
  print(fib(3))
  print(fib(4))
  print(fib(5))
}
```

```
test-fib.out
====================
1.000000000
1.000000000
2.000000000
3.000000000
5.000000000
8.000000000
```

```
test-duplicate1.y
====================
def addCap( e ) {
  e.cap == '10'
}

def main() {
  a == {||}
  b == deepCopy(a)
  b.cap == '0'
  addCap( a )
  print( a.cap )
  print( b.cap )
}
```

```
test-duplicate1.out
====================
10
0
```

```
test-dijkstra.y
====================
/* Given a list of vertices, return the index of the vertex with the
lowest
   value in the 'dist' attribute */
def minDistU( vertices ){
   minVertex == 0
   qCounter == 0
   minDistSoFar == INF
   forKeyValue( i, v, vertices ){
      thisDist == v.dist
      if( thisDist < minDistSoFar ) {
         minDistSoFar == thisDist
         minVertex == qCounter
      }
      qCounter == qCounter + 1
   }
   return( minVertex )
}

/* Run Dijkstra's algorithm from the source vertex to every other
vertex */
def dijkstras( G, source ){
  Q == []

  forKeyValue( label, v, v( G ) ){
      v.dist == INF
      v.prev == NULL
      append( v(G)[ label ], Q )
  }

  v(G)[source].dist == 0

  while( not isEmpty( Q ) ){
    u == remove( minDistU(Q), Q )

    forKeyValue( i, v, adj( G, u ) ) {
      alt == u.dist + e(G)[ edgeLabel( u, v ) ].length
      if ( alt < v.dist ) {
        v.dist == alt
        v.prev == u
      }
    }
  }
}

def main() {
  G == graph

  /* Construct the graph */
  e1 == addE( G, 'a', 'b' )
  e2 == addE( G, 'a', 'c' )
  e3 == addE( G, 'a', 'f' )
  e4 == addE( G, 'b', 'c' )
  e5 == addE( G, 'b', 'd' )
  e6 == addE( G, 'c', 'f' )
  e7 == addE( G, 'c', 'd' )
  e8 == addE( G, 'd', 'e' )
  e9 == addE( G, 'f', 'e' )
  e1.length == 7
  e2.length == 9
  e3.length == 14
  e4.length == 10
  e5.length == 15
  e6.length == 2
  e7.length == 11
  e8.length == 6
  e9.length == 9

  dijkstras( G, 'a' )

  print( 'Shortest distances' )
  print( 'a to a' )
  print( v( G ).a.dist )
  print( 'a to b' )
  print( v( G ).b.dist )
  print( 'a to c' )
  print( v( G ).c.dist )
  print( 'a to d' )
  print( v( G ).d.dist )
  print( 'a to e' )
  print( v( G ).e.dist )
  print( 'a to f' )
  print( v( G ).f.dist )

  /* Get the shortest path from vertex a to vertex e */
  u == v( G ).e
  path == [ 'e' ]
  while ( not isEqual( u.prev, NULL ) ) {
    u == v( G )[ u.prev.label ]
    push( u.label, path )
  }
  print( 'Shortest path from vertex a to vertex e' )
  while( size( path ) > 0 ) {
    print( pop( path ) )
  }
}
```

```
test-dijkstra.out
====================
Shortest distances
a to a
0.000000000
a to b
7.000000000
a to c
9.000000000
a to d
20.000000000
```

```
a to e
20.000000000
a to f
11.000000000
Shortest path from vertex a to vertex e
a
c
f
e

test-arith3.y
====================
def main() {
  a == 42
  a == a + 5
  print(a)
}

test-arith3.out
====================
47.000000000

test-arith2.y
====================
def main() {
  print(1 + 2 * 3 + 4)
}

test-arith2.out
====================
11.000000000

test-arith1.y
====================
def main() {
  print(39 + 3)
}

test-arith1.out
====================
42.000000000

test-add3.y
```

```
====================
def add(x, y) {
  return( x + y )
}

def main() {
  print( add( 'Hello', 'World' ) )
  print( add( 40, 2 ) )
}
```

```
test-add3.out
====================
HelloWorld
42.000000000
```

```
test-add2.y
====================
def add(x, y) {
  return( x + y )
}

def main() {
  print( add('Hello', 'World') )
}
```

```
test-add2.out
====================
HelloWorld
```

```
test-add1.y
====================
def add(x, y) {
  return( x + y )
}

def main() {
  print( add(17, 25) )
}
```

```
test-add1.out
====================
42.000000000
```

```
run-incompatible2.y
====================
def main() {
   print( '12' % '4' )
}
run-incompatible2.err

ERROR: Expected item of type:  0  Found item of type:  1
{| 'NULL' :== -1, 'Num' :== 0, 'String' :== 1, 'List' :== 2, 'Map'
:== 3, 'Graph' :== 4 |}

run-expr2.y
====================
def main(){
  print( 1 + 2 )
  print( 1 - 2 )
  print( 1 * 2 )
  print( 1 / 2 )
  print( 1 % 2 )
  print( 'foo' * 'string' )
}
run-expr2.err
3.000000000
-1.000000000
2.000000000
0.500000000
1.000000000

ERROR: Expected item of type:  0  Found item of type:  1
{| 'NULL' :== -1, 'Num' :== 0, 'String' :== 1, 'List' :== 2, 'Map'
:== 3, 'Graph' :== 4 |}

run-expr1.y
====================
def main(){
  1 + 2
  1 - 2
  1 * 2
  1 / 2
  1 % 2
  1 + 'string'
}
run-expr1.err
```

```
ERROR: Incompatible types
ERROR: left item of type:
0
ERROR: right item of type:
1
{| 'NULL' :== -1, 'Num' :== 0, 'String' :== 1, 'List' :== 2, 'Map'
:== 3, 'Graph' :== 4 |}

fail-return1.y
====================
def main() {
  return( 0 )
}
fail-return1.err
Fatal error: exception Failure("You may not return in main function")
fail-func9.err
Fatal error: exception Failure("illegal actual argument found int
expected bool in 42")
fail-func8.err
Fatal error: exception Failure("illegal actual argument found void
expected bool in bar()")

fail-func7.y
====================
def foo(a,b) {
}

def main() {
  foo(42, true)
  foo(42,true,false) /* Wrong number of arguments */
}
fail-func7.err
Fatal error: exception Failure("expecting 2 arguments in foo( 42.,
1., 0. )")

fail-func6.y
====================
def foo(a,b) {
}

def main() {
  foo(42, true)
  foo(42) /* Wrong number of arguments */
}
fail-func6.err
Fatal error: exception Failure("expecting 2 arguments in foo( 42. )")
fail-func5.err
Fatal error: exception Failure("illegal void local b in bar")

fail-func4.y
====================

def bar() {
}

def print() {
} /* Should not be able to define print */

def baz() {
}

def main() {
  print( 42 )
}
fail-func4.err
Fatal error: exception Failure("function print may not be defined")

fail-func3.y
====================

def bar() {
}

def remove() {
} /* Should not be able to define remove */

def baz() {
}

def main() {
  print( 42 )
}
fail-func3.err
Fatal error: exception Failure("function remove may not be defined")

fail-func2.y
```

```
====================
def foo( a,b,c ){
}

def bar(a,b,a){
}

def main() {
  print( 42 )
}
fail-func2.err
Fatal error: exception Failure("duplicate formal a in bar")

fail-func1.y
====================
def foo() {
}

def bar() {
}

def baz() {
}

def bar() {
} /* Error: duplicate function bar */

def main() {
  print( 42 )
}
fail-func1.err
Fatal error: exception Failure("duplicate function bar")

fail-forKey1.y
====================
def main() {
   forKey( 1, [1,2,3] ) {
   }
}
fail-forKey1.err
Fatal error: exception Failure("The first arg of ForEach must be an
Id")
```

```
fail-dead1.y
====================
def main() {
  i == 15
  return( i )
  i == 32 /* Error: code after a return */
}
fail-dead1.err
Fatal error: exception Failure("nothing may follow a return")

fail-assign5.y
====================
def main() {
  {} == {1}
}
fail-assign5.err
Fatal error: exception Failure("
 Illegal assignment!
 Can only assign to variables. Cannot assign to litteral
declarations, results of calls, or expressions
 Failed Expression: {  }=={ 1. }
")

fail-assign4.y
====================
def main() {
  {||} == {| 'key1' :== 1 |}
}
fail-assign4.err
Fatal error: exception Failure("
 Illegal assignment!
 Can only assign to variables. Cannot assign to litteral
declarations, results of calls, or expressions
 Failed Expression: {|  |}=={| key1 :== 1. |}
")

fail-assign3.y
====================
def main() {
  [] == [123]
}
fail-assign3.err
Fatal error: exception Failure("
 Illegal assignment!
 Can only assign to variables. Cannot assign to litteral
declarations, results of calls, or expressions
 Failed Expression: [  ]==[ 123. ]
")

fail-assign2.y
====================
def main() {
  '123' == '456'
}
fail-assign2.err
Fatal error: exception Failure("
 Illegal assignment!
 Can only assign to variables. Cannot assign to litteral
declarations, results of calls, or expressions
 Failed Expression: 123==456
")

fail-assign1.y
====================
def main() {
  10 == 3
}
fail-assign1.err
Fatal error: exception Failure("
 Illegal assignment!
 Can only assign to variables. Cannot assign to litteral
declarations, results of calls, or expressions
 Failed Expression: 10.==3.
")
```

8 Appendix

Anthony and David both put significant work into all of these files, and are joint authors for all the components.

Code:

```
Scanner.mll
(* Ocamllex scanner for YAGL *)

{ open Parser }

rule token = parse
  [' ' '\t' '\r'] { token lexbuf } (* Whitespace *)
| '\n' { NEWLINE }
| "/*" { m_comment lexbuf }
| "//" { s_comment lexbuf }
| "'" { QUOTE }
| '(' { LPAREN }
| ')' { RPAREN }
| '{' { LBRACE }
| '}' { RBRACE }
| "{|" { LMAP }
| "|}" { RMAP }
| '[' { LBRACKET }
| ']' { RBRACKET }
| ',' { COMMA }
| '.' { DOT }
| '+' { PLUS }
| '-' { MINUS }
| '*' { TIMES }
| '/' { DIVIDE }
| '%' { MOD }
| '=' { ASSIGN }
| ":=" { MAPASSIGN }
| "==" { EQ }
| "!=" { NEQ }
| '<' { LT }
| "<=" { LEQ }
| '>' { GT }
| ">=" { GEQ }
| "INF" { NUM(infinity) }
| "and" { AND }
| "or" { OR }
| "not" { NOT }
| "if" { IF }
| "else" { ELSE }
| "forKey" { FOREACH }
| "forKeyValue" { FORKV }
| "while" { WHILE }
| "NULL" { NULL }
| "true" { NUM(1.) }
| "false" { NUM(0.) }
| "def" { DEF }
| "return" { RETURN }
| "graph" { GRAPH }
| "digraph" { DIGRAPH }
| "typeOf" { TYPEOF }
| ['0'-'9']*'.'?['0'-'9']+ as lxm { NUM(float_of_string lxm) }
| ['0'-'9']+'.'?['0'-'9']* as lxm { NUM(float_of_string lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '~']* as lxm { ID(lxm) }
| "'"['a'-'z' 'A'-'Z' '0'-'9' ' ' '~']*"'" as lxm {
      STRING(String.sub lxm 1 (String.length lxm - 2)) }
| eof { EOF }
| _ as char { raise (Failure("The following character is an illegal YAGL character '"
^
             Char.escaped char ^ "'")) }

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

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

Parser.mly

```
/* Ocamlyacc parser for YAGL */

%{
open Ast
%}

%token LPAREN RPAREN LBRACE RBRACE COMMA DOT LMAP RMAP LBRACKET RBRACKET QUOTE
%token NEWLINE
%token PLUS MINUS TIMES DIVIDE MOD ASSIGN NOT MAPASSIGN
%token EQ NEQ LT LEQ GT GEQ AND OR INF
%token RETURN IF ELSE FOREACH FORKV WHILE NULL DEF
%token GRAPH DIGRAPH
%token TYPEOF
%token <float> NUM
%token <string> ID
%token <string> STRING
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE MOD
%right NOT
%nonassoc NEG
%left DOT LBRACKET RBRACKET
%left RPAREN

%start program
%type <Ast.program> program

%%

program:
  NEWLINE program { $2 }
| decls EOF { $1 }

decls:
  /* nothing */ { [] }
| decls fdecl { $2 :: $1 }

arb_newline:
  /* nothing */ { () }
| NEWLINE arb_newline { () }
```

```
fdecl:
  DEF ID LPAREN formals_opt RPAREN LBRACE NEWLINE stmt_list RBRACE NEWLINE
  arb_newline
       { { fname = $2; formals = $4; body = List.rev $8 } }

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

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

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

stmt:
  expr NEWLINE { Expr $1 }
| RETURN LPAREN RPAREN NEWLINE { Return Noexpr }
| RETURN LPAREN expr RPAREN NEWLINE { Return $3 }
| IF LPAREN expr RPAREN LBRACE NEWLINE stmt_list RBRACE NEWLINE %prec NOELSE
       { If($3, List.rev $7, []) }
| IF LPAREN expr RPAREN LBRACE NEWLINE stmt_list RBRACE NEWLINE ELSE LBRACE
  stmt_list RBRACE NEWLINE
       { If($3, List.rev $7, List.rev $12) }
| WHILE LPAREN expr RPAREN LBRACE NEWLINE stmt_list RBRACE NEWLINE
       { While($3, List.rev $7) }
| FOREACH LPAREN expr COMMA expr RPAREN LBRACE NEWLINE stmt_list RBRACE NEWLINE
       { Foreach($3, $5, List.rev $9) }
| FORKV LPAREN expr COMMA expr COMMA expr RPAREN LBRACE NEWLINE stmt_list RBRACE
  NEWLINE
       { Forkv($3, $5, $7, List.rev $11) }

| NEWLINE { Nothing }

expr:
| NUM { Num($1) }
| STRING { String($1) }
| LBRACKET actuals_opt RBRACKET { List($2) }
| LBRACE actuals_opt RBRACE { Set($2) }
| LMAP map_opt RMAP { Map($2) }
| GRAPH { Graph }
| DIGRAPH { Digraph }
| NULL { Null }
| TYPEOF LPAREN expr RPAREN { Typeof( $3 ) }
| 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 MOD expr { Binop($1, Mod, $3) }
| expr EQ     expr { Eqop($1, Equal, $3) }
| expr NEQ    expr { Eqop($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) }
| MINUS expr %prec NEG { Unop(Neg, $2) }
| NOT expr { Unop(Not, $2) }
| expr ASSIGN expr { Assign($1, $3) }
| ID LPAREN actuals_opt RPAREN { Call($1, $3) }
| expr LBRACKET expr RBRACKET { ListOrMapAccess($1, $3) }
/* NB: even though this is string access, we need to parse it as an ID because
   strings written with this syntax don't have quote marks around them. */
| expr DOT ID { MapStringAccess($1, $3) }
| ID { Id($1) }
| LPAREN expr RPAREN { $2 }

map_opt:
  /* nothing */ { [] }
| map_list { List.rev $1 }

map_list:
  STRING MAPASSIGN expr { [($1, $3)] }
| map_list COMMA STRING MAPASSIGN expr { ($3, $5) :: $1 }

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

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

MAKEFILE

```
# Make sure ocamlbuild can find opam-managed packages: first run
#
# eval `opam config env`

# Easiest way to build: using ocamlbuild, which in turn uses ocamlfind

.PHONY : yagl.native

yagl.native :
    ocamlbuild -use-ocamlfind -pkgs llvm,llvm.analysis -cflags -w,+a-4 \
        yagl.native

# "make clean" removes all generated files

.PHONY : clean
clean :
    ocamlbuild -clean
    rm -rf testall.log *.diff yagl scanner.ml parser.ml parser.mli
    rm -rf *.cmx *.cmi *.cmo *.cmx *.o

# More detailed: build using ocamlc/ocamlopt + ocamlfind to locate LLVM

OBJS = ast.cmx codegen.cmx parser.cmx scanner.cmx semant.cmx yagl.cmx

yagl : $(OBJS)
    ocamlfind ocamlopt -linkpkg -package llvm -package llvm.analysis $(OBJS) -o \
    yagl

scanner.ml : scanner.mll
    ocamllex scanner.mll

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

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

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

%.cmx : %.ml
    ocamlfind ocamlopt -c -package llvm $<

### Generated by "ocamldep *.ml *.mli" after building scanner.ml and parser.ml
ast.cmo :
ast.cmx :
codegen.cmo : ast.cmo
codegen.cmx : ast.cmx
yagl.cmo : semant.cmo scanner.cmo parser.cmi codegen.cmo ast.cmo
yagl.cmx : semant.cmx scanner.cmx parser.cmx codegen.cmx ast.cmx
parser.cmo : ast.cmo parser.cmi
parser.cmx : ast.cmx parser.cmi
scanner.cmo : parser.cmi
scanner.cmx : parser.cmx
semant.cmo : ast.cmo
semant.cmx : ast.cmx
parser.cmi : ast.cmo

# Building the tarball

TESTS = add1 arith1 arith2 arith3 fib for1 for2 func1 func2 func3    \
       func4 func5 func6 func7 func8 gcd2 gcd global1 global2 global3    \
       hello if1 if2 if3 if4 if5 local1 local2 ops1 ops2 var1 var2         \
       while1 while2

FAILS = assign1 assign2 assign3 dead1 dead2 expr1 expr2 for1 for2    \
       for3 for4 for5 func1 func2 func3 func4 func5 func6 func7 func8    \
       func9 global1 global2 if1 if2 if3 nomain return1 return2 while1    \
       while2

TESTFILES = $(TESTS:%=test-%.mc) $(TESTS:%=test-%.out) \
       $(FAILS:%=fail-%.mc) $(FAILS:%=fail-%.err)

TARFILES = ast.ml codegen.ml Makefile yagl.ml parser.mly README scanner.mll \
    semant.ml testall.sh $(TESTFILES:%=tests/%)

yagl-llvm.tar.gz : $(TARFILES)
    cd .. && tar czf yagl-llvm/yagl-llvm.tar.gz \
        $(TARFILES:%=yagl-llvm/%)
```

Codegen.ml

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

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

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

Detailed documentation on the OCaml LLVM library:

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

*)

module L = Llvm
module A = Ast

module StringMap = Map.Make(String);;

exception Uninitialized_var of string;;

module StringHash = Hashtbl.Make(struct
  type t = string
  let equal x y = x = y
  let hash = Hashtbl.hash end);;

let translate (functions) =
  let context = L.global_context () in
  let the_module = L.create_module context "YAGL"
  and i32_t = L.i32_type context
  and i8_t = L.i8_type context
  (* and i1_t = L.i1_type context *)
  and double_t = L.double_type context
  and void_t = L.void_type context
  and pointer_t = L.pointer_type
  and string_t = (L.array_type (L.i8_type context) 0)
  and struct_t = L.named_struct_type context "struct_t"
  and node_t = L.named_struct_type context "node_t"
  and graph_t = L.named_struct_type context "graph_t" in

  let zero = L.const_int (L.i32_type context) 0 and
      one = L.const_int (L.i32_type context) 1 and
      two = L.const_int (L.i32_type context) 2 and
      three = L.const_int (L.i32_type context) 3 and
      four = L.const_int (L.i32_type context) 4 and
      ten = L.const_int i32_t 10 and
      minus_one = L.const_int (L.i32_type context) (-1) in
  let num_type = L.const_int (L.i32_type context) 0 and
      str_type = L.const_int (L.i32_type context) 1 and
```

```
     lst_type = L.const_int (L.i32_type context) 2 and
     map_type = L.const_int (L.i32_type context) 3 and
     graph_type = L.const_int (L.i32_type context) 4 and
     null_type = L.const_int (L.i32_type context) (-1) in
let f_zero = L.const_float double_t 0.0 and
     f_one  = L.const_float double_t 1.0 in

(* Every data type in YAGL is represented with a single data type under the
     hood: a struct containing pointers to the possible YAGL types. No more than
     one of these pointers should be non-null at any point; the non-null type
     (if any) is the YAGL type that the struct actually represents. *)
ignore (L.struct_set_body struct_t
     [| pointer_t double_t;  (* number type *)
     pointer_t string_t;  (* string type *)
     pointer_t node_t;  (* list type *)
     pointer_t node_t;  (* map type *)
     pointer_t graph_t  (* graph type *)
     |]
false);

(* Node in the style of a linked list node for representing lists, sets, and
     maps *)
ignore (L.struct_set_body node_t
     [| pointer_t string_t;  (* Key for maps; null for lists *)
     pointer_t struct_t;  (* Data for this entry *)
     pointer_t node_t  (* Pointer to the next node *)
     |]
false);

(* Graph representation *)
ignore (L.struct_set_body graph_t
     [| pointer_t node_t;  (* Vertices *)
     pointer_t node_t  (* Edges *)
     |]
false);

(* C's printf *)
let printf_t = L.var_arg_function_type i32_t [| pointer_t i8_t |] in
let printf_func = L.declare_function "printf" printf_t the_module in

(* C's abort *)
(* TODO: delete if this is not actually used anywhere *)
let abort_t = L.var_arg_function_type void_t [||] in
let abort_func = L.declare_function "abort" abort_t the_module in

(* C's strcmp *)
let strcmp_t = L.var_arg_function_type i32_t [| pointer_t i8_t; pointer_t i8_t |] in
let strcmp_func = L.declare_function "strcmp" strcmp_t the_module in

(* C's strcat *)
```

```
let strcat_t = L.var_arg_function_type i32_t [| pointer_t i8_t; pointer_t i8_t |] in
let strcat_func = L.declare_function "strcat" strcat_t the_module in

(* C's strcpy *)
let strcpy_t = L.var_arg_function_type i32_t [| pointer_t i8_t; pointer_t i8_t |] in
let strcpy_func = L.declare_function "strcpy" strcpy_t the_module in

(* C's strlen *)
let strlen_t = L.var_arg_function_type i32_t [| pointer_t i8_t |] in
let strlen_func = L.declare_function "strlen" strlen_t the_module in

(* Define my own funciton *)
let myfunc_t = L.var_arg_function_type double_t [| pointer_t double_t |] in
let myfunc_func = L.define_function "myfunc" myfunc_t the_module in
let myfunc_bb = L.entry_block myfunc_func in
let myfunc_builder = L.builder_at_end context myfunc_bb in
     let a_ptr = L.param myfunc_func 0 in
     let b_ptr = L.param myfunc_func 1 in
     let a_real = (L.build_load a_ptr "a_load" myfunc_builder ) in
     let a_ret = L.build_fadd a_real f_one "myf_add" myfunc_builder in
     ignore( L.build_ret a_ret myfunc_builder );
 (*   ignore(L.position_at_end myfunc_bb myfunc_builder); *)


(* As the default type of any variable in the IR is a struct, the type of any
     input to a function is a struct pointer. Every function returns a struct
     in the IR, except for the main function, which returns void. *)
let function_decls =
     let rec list_of_n_struct_ptrs n =
     if n = 0 then []
     else pointer_t struct_t :: (list_of_n_struct_ptrs (n - 1)) in
     let array_of_n_struct_ptrs n = Array.of_list (list_of_n_struct_ptrs n) in

     let function_decl m fdecl =
     let name = fdecl.A.fname in
     let return_type =
     if name = "main" then void_t
     else (pointer_t struct_t) in
     StringMap.add name
     (L.define_function name
     (L.function_type
            return_type (array_of_n_struct_ptrs (List.length fdecl.A.formals)))
     the_module,
     fdecl) m in
     List.fold_left function_decl StringMap.empty functions in

let build_function_body fdecl =
     let (the_function, _) = StringMap.find fdecl.A.fname function_decls in

     (* The main builder for the function body *)
```

```
let builder = L.builder_at_end context (L.entry_block the_function) in

(* Global strings for printing variables and error messages *)
(* TODO: move out of build_function_body; it doesn't make sense here *)
let int_format_str =
L.build_global_stringptr "%d\n" "fmt_int" builder in
let float_format_str =
L.build_global_stringptr "%.9f\n" "fmt_float" builder in
let string_format_str =
L.build_global_stringptr "%s\n" "fmt_string" builder in
let bad_index_str =
L.build_global_stringptr "ERROR: Bad list index" "bad_index" builder in
let diff_types_str =
L.build_global_stringptr "ERROR: Incompatible types" "diff_types" builder in
let invalid_print_str =
L.build_global_stringptr "ERROR: Invalid print type" "print_types" builder in
let type_check_format_str =
L.build_global_stringptr ("\nERROR: Expected item of type:  %d  " ^
"Found item of type:  %d\n{| 'NULL' := -1, 'Num' := 0, 'String' := 1," ^
" 'List' := 2, 'Map' := 3, 'Graph' := 4 |}\n") "type_err_string" builder
in
let left_type_str =
L.build_global_stringptr "ERROR: left item of type:" "non_lst" builder in
let right_type_str =
L.build_global_stringptr "ERROR: right item of type:" "non_lst" builder in
let description_type_str =
L.build_global_stringptr ("{| 'NULL' := -1, 'Num' := 0, 'String' := 1, " ^
"'List' := 2, 'Map' := 3, 'Graph' := 4 |}") "description" builder in
let here_str =
L.build_global_stringptr "HERE" "here" builder in
let null_str =
L.build_global_stringptr "NULL" "NULL" builder in

let print_here builder =
ignore( L.build_call printf_func [| string_format_str; here_str |]
"here" builder )
in

(* In YAGL, when a variable is defined anywhere in a function, it is
accessible at any point in the control flow afterwards, even if it was
defined in, say, an if-else block. To make this possible in LLVM, we
create a basic block for mallocting new memory for variables, which gets
run first. Any time we use build_malloc, we also use the builder
rather than the main builder. *)
let malloc_bb = L.append_block context "malloc_vars" the_function in
let malloc_builder = L.builder_at_end context malloc_bb in
ignore (L.build_br malloc_bb builder);

(* The basic block for function statements, which gets run after the
variable malloction basic block in the IR. The break from the malloction
block to this block is set at the very end, after we are sure that no
more variables can be defined. *)
let stmts_bb = L.append_block context "stmts" the_function in
L.position_at_end stmts_bb builder;

(* TODO: this function was copied over and is still a bit mysterious.
Investigate and either remove or write a better comment. *)
let add_terminal builder f =
match L.block_terminator (L.insertion_block builder) with
Some _ -> ()
| None -> ignore (f builder)
in

(* Helper functions to retrieve data pointers from a struct. *)
let extract_struct_index s i builder =
let struct_elem =
L.build_in_bounds_gep s [| zero; i |] "struct_elem" builder in
L.build_load struct_elem "elem_ptr" builder
in
let extract_num s builder = extract_struct_index s zero builder in
let extract_string s builder = extract_struct_index s one builder in
let extract_list s builder = extract_struct_index s two builder in
let extract_map s builder = extract_struct_index s three builder in
let extract_graph s builder = extract_struct_index s four builder in

let extract_num_val s builder =
L.build_load (extract_num s builder) "num_val" builder
in

let extract_list_val s builder =
L.build_load (extract_list s builder) "num_val" builder
in
(* Helper functions to retrieve data pointers from a node *)
let extract_node_string n builder = extract_struct_index n zero  builder in
let extract_node_struct n builder = extract_struct_index n one   builder in
let extract_node_node   n builder = extract_struct_index n two builder in

let extract_graph_v g builder = extract_struct_index g zero builder in
let extract_graph_e g builder = extract_struct_index g one builder in

(* Helper function which takes a struct contianing a number and returns an
int value 0 or 1 if it represents a true or false value *)

let is_true s builder =
let num_val = extract_num_val s builder in
L.build_fcmp L.Fcmp.One num_val f_zero "is_true" builder
in

let valid_struct_index s builder =
let is_num =
L.build_is_not_null (extract_num s builder) "is_num" builder in
let is_string =
L.build_is_not_null (extract_string s builder) "is_string" builder in
let is_list =
L.build_is_not_null (extract_list s builder) "is_list" builder in
let is_map =
L.build_is_not_null (extract_map s builder) "is_map" builder in
let is_graph =
L.build_is_not_null (extract_graph s builder) "is_graph" builder in
let result = L.build_malloc i32_t "result" builder in

let merge_bb = L.append_block context "merge_valid_struct" the_function in

let then_graph_bb = L.append_block context "then_graph"
the_function in
let then_graph_builder = L.builder_at_end context then_graph_bb in
ignore (L.build_store four result then_graph_builder);
ignore (L.build_br merge_bb then_graph_builder);

let not_graph_bb = L.append_block context "not_graph" the_function in
let not_graph_builder = L.builder_at_end context not_graph_bb in
ignore (L.build_store minus_one result not_graph_builder);
ignore (L.build_br merge_bb not_graph_builder);

let then_map_bb = L.append_block context "then_map" the_function in
let then_map_builder = L.builder_at_end context then_map_bb in
ignore (L.build_store three result then_map_builder);
ignore (L.build_br merge_bb then_map_builder);

let not_map_bb = L.append_block context "not_map" the_function in
let not_map_builder = L.builder_at_end context not_map_bb in
ignore (L.build_cond_br is_graph then_graph_bb not_graph_bb not_map_builder);

let then_list_bb = L.append_block context "then_list" the_function in
let then_list_builder = L.builder_at_end context then_list_bb in
ignore (L.build_store two result then_list_builder);
ignore (L.build_br merge_bb then_list_builder);

let not_list_bb = L.append_block context "not_list" the_function in
let not_list_builder = L.builder_at_end context not_list_bb in
ignore (L.build_cond_br is_map then_map_bb not_map_bb not_list_builder);

let then_str_bb = L.append_block context "then_str" the_function in
let then_str_builder = L.builder_at_end context then_str_bb in
ignore (L.build_store one result then_str_builder);
ignore (L.build_br merge_bb then_str_builder);

let not_str_bb = L.append_block context "not_str" the_function in
let not_str_builder = L.builder_at_end context not_str_bb in
ignore (L.build_cond_br is_list then_list_bb not_list_bb not_str_builder);
```

```
let then_num_bb = L.append_block context "then_num" the_function in
let then_num_builder = L.builder_at_end context then_num_bb in
ignore (L.build_store zero result then_num_builder);
ignore (L.build_br merge_bb then_num_builder);

let not_num_bb = L.append_block context "not_num" the_function in
let not_num_builder = L.builder_at_end context not_num_bb in
ignore (L.build_cond_br is_string then_str_bb not_str_bb not_num_builder);

ignore (L.build_cond_br is_num then_num_bb not_num_bb builder);
L.position_at_end merge_bb builder;
L.build_load result "result" builder;
in

(* Helper function to build an empty struct containing null pointers. *)
let build_empty_struct builder n =
let empty_struct = L.build_malloc struct_t n builder in
let struct_double = L.build_in_bounds_gep empty_struct [| zero; zero |]
       "struct_double" builder and
null_double = L.const_pointer_null (pointer_t double_t) and
struct_string = L.build_in_bounds_gep empty_struct [| zero; one |]
       "struct_string" builder and
null_string = L.const_pointer_null (pointer_t string_t) and
struct_list = L.build_in_bounds_gep empty_struct [| zero; two |]
       "struct_list" builder and
null_list = L.const_pointer_null (pointer_t node_t) and
struct_map = L.build_in_bounds_gep empty_struct [| zero; three |]
       "struct_map" builder and
null_map = L.const_pointer_null (pointer_t node_t) and
struct_graph = L.build_in_bounds_gep empty_struct [| zero; four |]
       "struct_graph" builder and
null_graph = L.const_pointer_null (pointer_t graph_t) in
ignore (L.build_store null_double struct_double builder);
ignore (L.build_store null_string struct_string builder);
ignore (L.build_store null_list struct_list builder);
ignore (L.build_store null_map struct_map builder);
ignore (L.build_store null_graph struct_graph builder);
empty_struct
in

(* Helper function to build an empty node containing null pointers. *)
let build_empty_node builder n =
let empty_node = L.build_malloc node_t n builder in
let node_str = L.build_in_bounds_gep empty_node [| zero; zero |]
       "node_str" builder and
null_str = L.const_pointer_null (pointer_t string_t) and
node_struct = L.build_in_bounds_gep empty_node [| zero; one |]
       "node_struct" builder and
null_struct = L.const_pointer_null (pointer_t struct_t) and
```

```
node_node = L.build_in_bounds_gep empty_node [| zero; two |]
       "node_node" builder and
null_node = L.const_pointer_null (pointer_t node_t) in
ignore (L.build_store null_str node_str builder);
ignore (L.build_store null_struct node_struct builder);
ignore (L.build_store null_node node_node builder);
empty_node
in

(* Helper function to build an empty graph. *)
let build_empty_graph builder n =
let empty_graph = L.build_malloc graph_t n builder in
let v = L.build_in_bounds_gep empty_graph [| zero; zero |]
       "v" builder and
e = L.build_in_bounds_gep empty_graph [| zero; one |]
       "e" builder in
ignore (L.build_store (build_empty_node builder "v_node") v builder);
ignore (L.build_store (build_empty_node builder "e_node") e builder);
empty_graph
in

(* Helper function to create a struct containing a number. *)
let construct_str str_ptr builder =
let str_struct = build_empty_struct builder "num_struct" in
let struct_str = L.build_in_bounds_gep str_struct [| zero; one |]
"struct_num" builder in
ignore (L.build_store str_ptr struct_str builder);
str_struct
in

(* Helper function to create a struct containing a number. *)
let construct_num n builder =
let num_struct = build_empty_struct builder "num_struct" in
let struct_num = L.build_in_bounds_gep num_struct [| zero; zero |]
"struct_num" builder in
let num_ptr = L.build_malloc double_t "num_ptr" builder in
ignore (L.build_store n num_ptr builder);
ignore (L.build_store num_ptr struct_num builder);
num_struct
in

(* Helper function to create a struct containing a graph. *)
let construct_graph builder =
let graph_struct = build_empty_struct builder "graph_struct" in
let struct_graph = L.build_in_bounds_gep graph_struct [| zero; four |]
"struct_graph" builder in
ignore (L.build_store (build_empty_graph builder "graph") struct_graph
builder);
graph_struct
in
```

```
(* Initialize the local variables as the variables that were passed into the
function. We store local variables in a hash map from the name to a
struct pointer; the pointer is necessary to make sure that we handle
variable reassignment properly. *)
let local_vars =
let add_formal m n p =
L.set_value_name n p;
StringHash.add m n p;
m in
List.fold_left2 add_formal (StringHash.create 0) fdecl.A.formals
(Array.to_list (L.params the_function))
in

let print_struct builder s =
let merge_bb = L.append_block context "merge_print" the_function in

(* Build print num and print string blocks *)
let print_num_bb = L.append_block context "print_num" the_function in
let print_str_bb = L.append_block context "print_str" the_function in
let print_null_bb = L.append_block context "print_null" the_function in

let null_builder = L.builder_at_end context print_null_bb in
ignore( L.build_call printf_func [| string_format_str; null_str |]
       "printf" null_builder );
ignore( L.build_br merge_bb null_builder);

let num_builder = L.builder_at_end context print_num_bb in
let num = L.build_load ( extract_num s num_builder ) "num" num_builder in
ignore( L.build_call printf_func [| float_format_str; num |] "printf"
num_builder; );
ignore (L.build_br merge_bb num_builder);

let str_builder = L.builder_at_end context print_str_bb in
ignore( L.build_call printf_func [| string_format_str;
( extract_string s str_builder ) |] "printf" str_builder; );
ignore (L.build_br merge_bb str_builder);

(* Build invalid print type block *)
let invalid_type_bb = L.append_block context "invalid_print"
the_function in
let invalid_builder = L.builder_at_end context invalid_type_bb in
ignore( L.build_call printf_func
[| string_format_str; invalid_print_str |] "printf" invalid_builder );
ignore( L.build_call abort_func [||] "" invalid_builder);
ignore (L.build_br merge_bb invalid_builder);

(* Determine what type this struct is *)
let struct_type = valid_struct_index s builder in
```

```
(* Construct switch statement between blocks *)
let switch = L.build_switch struct_type invalid_type_bb 3 builder in
L.add_case switch minus_one print_null_bb; (* Print null string *)
L.add_case switch zero print_num_bb; (* Print like double *)
L.add_case switch one print_str_bb; (* Print like string *)

L.position_at_end merge_bb builder;
in

(* helper function to compare types of two structs returns type of structs
if same else raises exception *)
let type_comparison s1 s2 builder =
let merge_bb = L.append_block context "merge_type_check" the_function in

(* Determine if two structs can be operands to a binary operator *)
let s1_index   = valid_struct_index s1 builder in
let s2_index   = valid_struct_index s2 builder in
let is_null_s1 = L.build_icmp L.Icmp.Eq s1_index minus_one "is_null"
builder in
let is_null_s2 = L.build_icmp L.Icmp.Eq s2_index minus_one "is_null"
builder in
let either_null= L.build_or is_null_s1 is_null_s2 "either_null" builder in
let diff_type  = L.build_icmp L.Icmp.Ne s1_index s2_index "is_diff"
builder in
let is_invalid = L.build_or either_null diff_type "is_invalid" builder in

(* If different types print they are different then throw *)
let then_bb = L.append_block context "invalid_block" the_function in
let then_builder = L.builder_at_end context then_bb in
ignore( L.build_call printf_func [| string_format_str; diff_types_str |]
"printf" then_builder );
ignore( L.build_call printf_func [| string_format_str; left_type_str |]
"expected_type" then_builder );
ignore( L.build_call printf_func [| int_format_str; s1_index |]
"expected_type" then_builder );
ignore( L.build_call printf_func [| string_format_str; right_type_str |]
"found_type" then_builder );
ignore( L.build_call printf_func [| int_format_str; s2_index |]
"found_type" then_builder );
ignore( L.build_call printf_func [| string_format_str;
description_type_str |] "found_type" then_builder );
ignore (L.build_call abort_func [||] "" then_builder);
ignore (L.build_br merge_bb then_builder);

(* Else store the type in the return pointer *)
let else_bb = L.append_block context "valid_block" the_function in
let else_builder = L.builder_at_end context else_bb in
ignore(L.build_br merge_bb else_builder);

(* Construct the if statement *)
```

```
ignore( L.build_cond_br is_invalid then_bb else_bb builder);
L.position_at_end merge_bb builder;

(* Load return value *)
s1_index;
in

(* Checks if the type represented by an llvm number matches the correct type
also represented by an llvm number. If not throws a useful error on the
offending types *)
let check_index_type this_type correct_type builder =

let is_not_num = L.build_icmp L.Icmp.Ne this_type correct_type
"is_not_correct" builder in
let merge_bb = L.append_block context "merge_check" the_function in

let then_bb = L.append_block context "type_check_throw" the_function in
let then_builder = L.builder_at_end context then_bb in
ignore( L.build_call printf_func [| type_check_format_str; correct_type;
this_type |] "expected_type" then_builder );
ignore( L.build_call abort_func [||] "" then_builder);
ignore (L.build_br merge_bb then_builder);

ignore( L.build_cond_br is_not_num then_bb merge_bb builder);
L.position_at_end merge_bb builder;
in
(* Checks if struct is of a specific type. Also returns the struct type *)
let check_struct_type s correct_type builder =
let this_type = valid_struct_index s builder in
ignore( check_index_type this_type correct_type builder );
this_type;
in

(* Takes two structs; returns a struct with 1. or 0. indicating equality *)
let is_equal s1 s2 builder =

let s1_type = valid_struct_index s1 builder and (* s1_type is int *)
s2_type = valid_struct_index s2 builder in (* s2_type is int *)
let is_diff_type = L.build_intcast
( L.build_icmp L.Icmp.Ne s1_type s2_type "is_diff" builder) i32_t
"cast_int" builder in (* is_diff_type is unsigned int *)
(* ten_diff is type int *)
let ten_diff = L.build_mul is_diff_type ten "mul" builder in
(* this way if s1 and s2 are the same type then match is s1_type and if
they are diff types switch_match is s1_type + 10 *)
let switch_match = L.build_add ten_diff s1_type "add" builder in
let ret_ptr = L.build_malloc (pointer_t struct_t) "ret_ptr" builder in

let merge_bb = L.append_block context "merge_equal" the_function in
```

```
(* Default Not Equal Block *)
let not_equal_bb = L.append_block context "not_equal" the_function in
let ne_builder = L.builder_at_end context not_equal_bb in
ignore(L.build_store ( construct_num f_zero ne_builder ) ret_ptr
ne_builder);
ignore (L.build_br merge_bb ne_builder);

(* Null Equality Block - Note: all nulls are equal so always return true *)
let null_bb = L.append_block context "null_eq" the_function in
let null_builder = L.builder_at_end context null_bb in
ignore(L.build_store ( construct_num f_one null_builder ) ret_ptr
null_builder);
ignore (L.build_br merge_bb null_builder);

(* Number Equality Block *)
let double_bb = L.append_block context "double_eq" the_function in
let double_builder = L.builder_at_end context double_bb in
let num1 = extract_num_val s1 double_builder and
       num2 = extract_num_val s2 double_builder in
let outcome = L.build_fcmp L.Fcmp.Oeq num1 num2 "eq" double_builder in
let f_outcome = L.build_uitofp outcome double_t "fp" double_builder in
ignore(L.build_store (construct_num f_outcome double_builder) ret_ptr
double_builder);
ignore (L.build_br merge_bb double_builder);

(* String Equality Block *)
let string_bb = L.append_block context "string_eq" the_function in
let string_builder = L.builder_at_end context string_bb in
let str1 = extract_string s1 string_builder and
       str2 = extract_string s2 string_builder in
(* Get pointers to first char in either string  *)
let char_ptr1 = L.build_in_bounds_gep str1 [| zero; zero |] "char_ptr"
       string_builder and
       char_ptr2 = L.build_in_bounds_gep str2 [| zero; zero |] "char_ptr"
       string_builder in
let strcmp_val = L.build_call strcmp_func [| char_ptr1; char_ptr2 |]
"str_eq" string_builder in (* Returns 0 if equal so need to flip *)
let outcome = L.build_icmp L.Icmp.Eq strcmp_val zero "str_flip"
string_builder in
let f_outcome = L.build_uitofp outcome double_t "fp" string_builder in
ignore( L.build_store (construct_num f_outcome string_builder) ret_ptr
string_builder);
ignore (L.build_br merge_bb string_builder);

let switch = L.build_switch switch_match not_equal_bb 3 builder in
L.add_case switch minus_one null_bb;   (* Go to Null Equality   Block *)
L.add_case switch zero     double_bb; (* Go to Number Equality Block *)
L.add_case switch one      string_bb; (* Go to String Equality Block *)

L.position_at_end merge_bb builder;
```

```
L.build_load ret_ptr "data" builder;
in

let list_append l s builder =
let node_ptr = L.build_malloc (pointer_t node_t) "node_ptr"
builder in
ignore (L.build_store (extract_list l builder) node_ptr builder);

let pred_bb = L.append_block context "for_each_while" the_function in
ignore (L.build_br pred_bb builder);

let body_bb = L.append_block context "list_iter_body" the_function in
let body_builder = L.builder_at_end context body_bb in
let node = L.build_load node_ptr "node" body_builder in
let next_node = extract_node_node node body_builder in
ignore (L.build_store next_node node_ptr body_builder);
ignore (L.build_br pred_bb body_builder);

let pred_builder = L.builder_at_end context pred_bb in

let node = L.build_load node_ptr "node_ptr" pred_builder in
let node_struct = extract_node_struct node pred_builder in
let bool_val = L.build_is_not_null node_struct "bool_val" pred_builder in

let merge_bb = L.append_block context "merge_for_each" the_function in
ignore (L.build_cond_br bool_val body_bb merge_bb pred_builder);
L.position_at_end merge_bb builder;

let node_struct_ptr = L.build_in_bounds_gep node [| zero; one |]
"node_struct_ptr" builder in
let node_node = L.build_in_bounds_gep node [| zero; two |]
"node_node" builder in
let next_node = (build_empty_node builder "next_node") in
ignore (L.build_store s node_struct_ptr builder);
ignore (L.build_store next_node node_node builder);
in

(* Given a pointer to the first node of a list, and a pointer to an integer
index i, iterate over the list and retrieve the ith element. *)
let list_iter_node builder node_ptr i_ptr =
let pred_bb = L.append_block context "list_while" the_function in
ignore (L.build_br pred_bb builder);

let body_bb = L.append_block context "list_iter_body" the_function in
let body_builder = L.builder_at_end context body_bb in
(* Set the node pointer to the next node and the index pointer to i-1 *)
let node = L.build_load node_ptr "node" body_builder in
let next_node = extract_node_node node body_builder in
let i = L.build_load i_ptr "i" body_builder in
let tmp_i = L.build_sub i one "tmp_i" body_builder in
```

```
ignore (L.build_store next_node node_ptr body_builder);
ignore (L.build_store tmp_i i_ptr body_builder);
ignore (L.build_br pred_bb body_builder);

let pred_builder = L.builder_at_end context pred_bb in
let bool_val =
(* End iteration if i is 0 *)
let i = L.build_load i_ptr "i" pred_builder in
L.build_icmp L.Icmp.Ne i zero "not_zero" pred_builder 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.position_at_end merge_bb builder;

L.build_load node_ptr "node" builder;
in

let list_iter builder node_ptr i_ptr =
let node = list_iter_node builder node_ptr i_ptr in
extract_node_struct node builder
in

(* Function for list access *)
(* Takes a list node and a float *)
let access_list builder node num =
(* Helper functions to ensure that the index is a nonnegative integer *)
let is_int = L.build_fcmp L.Fcmp.Oeq
(L.build_frem num (L.const_float double_t 1.0) "mod_one" builder)
(L.const_float double_t 0.0)
"is_int" builder in
let is_nonneg = L.build_fcmp L.Fcmp.Oge
num
(L.const_float double_t 0.0)
"is_nonneg" builder in
let int_num = L.build_fptoui (L.build_fadd num
(L.const_float double_t 0.5)
"num_mod" builder) i32_t "int_num" builder in
let int_num_ptr = L.build_malloc i32_t "int_num_ptr" builder in
ignore (L.build_store int_num int_num_ptr builder);

let bool_val = L.build_and is_int is_nonneg "bool_val" builder in
let merge_bb = L.append_block context "merge_index_check" the_function in
let good_index_bb = L.append_block context "good_index" the_function in
let bad_index_bb = L.append_block context "bad_index" the_function in

(* Pointer to the data struct that will be returned (or set to null if the
index is bad) *)
let data_ptr = L.build_malloc (pointer_t struct_t) "data_ptr"
builder in
```

```
let node_ptr = L.build_malloc (pointer_t node_t) "node_ptr"
builder in
ignore (L.build_store node node_ptr builder);

let good_index_builder = L.builder_at_end context good_index_bb in
ignore (L.build_store (list_iter good_index_builder node_ptr
int_num_ptr) data_ptr good_index_builder);
ignore (L.build_br merge_bb good_index_builder);

let bad_index_builder = L.builder_at_end context bad_index_bb in
ignore (L.build_call printf_func [| string_format_str; bad_index_str |]
"printf" bad_index_builder);
ignore (L.build_call abort_func [||] "" bad_index_builder);
ignore (L.build_br merge_bb bad_index_builder);

ignore (L.build_cond_br bool_val good_index_bb bad_index_bb builder);
L.position_at_end merge_bb builder;
L.build_load data_ptr "data" builder;
in

let map_iter builder node_ptr str_ptr =
let pred_bb = L.append_block context "map_while" the_function in
let pred_continue_bb = L.append_block context "map_while_continue"
the_function in
let map_end_bb = L.append_block context "map_end" the_function in
let merge_bb = L.append_block context "merge" the_function in

ignore (L.build_br pred_bb builder);

let body_bb = L.append_block context "map_iter_body" the_function in
let body_builder = L.builder_at_end context body_bb in
(* Set the node pointer to the next node and the index pointer to i-1 *)
let node = L.build_load node_ptr "node" body_builder in
let next_node = extract_node_node node body_builder in
ignore (L.build_store next_node node_ptr body_builder);
ignore (L.build_br pred_bb body_builder);

let pred_builder = L.builder_at_end context pred_bb in

let node = L.build_load node_ptr "node_ptr" pred_builder in

let map_end_builder = L.builder_at_end context map_end_bb in
(* Add a node to the map and return an empty struct *)
let node_string_ptr = L.build_in_bounds_gep node [| zero; zero |]
"node_string_ptr" map_end_builder in
let node_struct_ptr = L.build_in_bounds_gep node [| zero; one |]
"node_struct_ptr" map_end_builder in
let node_node_ptr = L.build_in_bounds_gep node [| zero; two |]
"node_node_ptr" map_end_builder in
let new_struct = build_empty_struct map_end_builder "new_struct" in
let new_node = build_empty_node map_end_builder "new_node" in
ignore (L.build_store str_ptr node_string_ptr map_end_builder);
ignore (L.build_store new_struct node_struct_ptr map_end_builder);
ignore (L.build_store new_node node_node_ptr map_end_builder);
ignore (L.build_br merge_bb map_end_builder);

let node_struct = extract_node_struct node pred_builder in
let map_end = L.build_is_null node_struct "map_end" pred_builder in
ignore (L.build_cond_br map_end map_end_bb pred_continue_bb pred_builder);
L.position_at_end pred_continue_bb pred_builder;

let node_str = extract_node_string node pred_builder in
let node_str_begin = L.build_in_bounds_gep node_str [| zero; zero |]
"node_str_begin" pred_builder in (* Pointer to first char in str *)
(* Pointer to the fist char of comparison str *)
let str_begin = L.build_in_bounds_gep str_ptr [| zero; zero |]
"str_begin" pred_builder in
let strcmp_val = L.build_call strcmp_func [| node_str_begin; str_begin |]
"strcmp" pred_builder in
let bool_val = L.build_icmp L.Icmp.Ne strcmp_val zero "str_eq"
pred_builder in

ignore (L.build_cond_br bool_val body_bb merge_bb pred_builder);
L.position_at_end merge_bb builder;

let node = L.build_load node_ptr "node" builder in
extract_node_struct node builder
in

(* Function for map access *)
let access_map builder node str_ptr =
let node_ptr = (L.build_malloc (pointer_t node_t) "node_ptr"
builder) in
ignore (L.build_store node node_ptr builder);

map_iter builder node_ptr str_ptr
in

let access_list_or_map builder lm i =
let struct_index = valid_struct_index i builder in

let bool_val = L.build_icmp L.Icmp.Eq struct_index zero "is_num"
builder in
let merge_bb = L.append_block context "merge_access" the_function in
let access_list_bb = L.append_block context "access_list" the_function in
let access_map_bb = L.append_block context "access_map" the_function in

let data_ptr = L.build_malloc (pointer_t struct_t) "data_ptr"
builder in
```

```
let access_list_builder = L.builder_at_end context access_list_bb in
let lst = extract_list lm access_list_builder in
let num = extract_num_val i access_list_builder in
ignore (L.build_store (access_list access_list_builder lst num) data_ptr
access_list_builder);
ignore (L.build_br merge_bb access_list_builder);

let access_map_builder = L.builder_at_end context access_map_bb in
let map = extract_map lm access_map_builder in
let key_ptr = extract_string i access_map_builder in
ignore (L.build_store (access_map access_map_builder map key_ptr)
data_ptr access_map_builder);
ignore (L.build_br merge_bb access_map_builder);

ignore (L.build_cond_br bool_val access_list_bb access_map_bb builder);
L.position_at_end merge_bb builder;
L.build_load data_ptr "data" builder
in

let remove_map str_struct node_struct builder =
let node_ptr = L.build_malloc (pointer_t node_t) "node_ptr" builder in
let map = extract_map node_struct builder in
let str_ptr = extract_string str_struct builder in
ignore (L.build_store map node_ptr builder);

let prev_node_next_ptr_ptr = L.build_malloc (pointer_t node_t)
"prev_node_next_ptr_ptr" builder in
ignore (L.build_store map prev_node_next_ptr_ptr builder);

let pred_bb = L.append_block context "map_while" the_function in
let pred_continue_bb = L.append_block context "map_while_continue"
the_function in
let merge_bb = L.append_block context "merge" the_function in

ignore (L.build_br pred_bb builder);

let body_bb = L.append_block context "map_iter_body" the_function in
let body_builder = L.builder_at_end context body_bb in
(* Set the node pointer to the next node and the index pointer to i-1 *)
let node = L.build_load node_ptr "node" body_builder in
let next_node = extract_node_node node body_builder in
ignore (L.build_store next_node prev_node_next_ptr_ptr body_builder);
ignore (L.build_store next_node node_ptr body_builder);
ignore (L.build_br pred_bb body_builder);

let pred_builder = L.builder_at_end context pred_bb in

let node = L.build_load node_ptr "node_ptr" pred_builder in
let node_struct = extract_node_struct node pred_builder in
let map_end = L.build_is_null node_struct "map_end" pred_builder in
```

```
ignore (L.build_cond_br map_end merge_bb pred_continue_bb pred_builder);
L.position_at_end pred_continue_bb pred_builder;

let node_str = extract_node_string node pred_builder in
let node_str_begin = L.build_in_bounds_gep node_str [| zero; zero |]
"node_str_begin" pred_builder in (* Pointer to first char in str *)
(* Pointer to the fist char of comparison str *)
let str_begin = L.build_in_bounds_gep str_ptr [| zero; zero |]
"str_begin" pred_builder in
let strcmp_val = L.build_call strcmp_func [| node_str_begin; str_begin |]
"strcmp" pred_builder in
let bool_val = L.build_icmp L.Icmp.Ne strcmp_val zero "str_eq"
pred_builder in
ignore (L.build_cond_br bool_val body_bb merge_bb pred_builder);

L.position_at_end merge_bb builder;

let node = L.build_load node_ptr "node" builder in
let next_node_ptr = extract_node_node node builder in
let next_node = L.build_load next_node_ptr "next_node" builder in
let prev_node_next_ptr = L.build_load prev_node_next_ptr_ptr
"prev_node_next_ptr" builder in
ignore (L.build_store next_node prev_node_next_ptr builder);
in

(* Takes two structs, type checks to see if they are number and list
respectively then removes the item represented by the number in i *)
let remove_list i lst builder =
let node = extract_list lst builder in
let node_ptr = (L.build_malloc (pointer_t node_t) "node_ptr"
builder) in
let node_ptr2 = (L.build_malloc (pointer_t node_t) "node_ptr"
builder) in
ignore (L.build_store node node_ptr builder);
ignore (L.build_store node node_ptr2 builder);

(* Get node to be removed TODO error check out of bounds  *)
let i_val = extract_num_val i builder in
let i_int = L.build_fptoui i_val i32_t "int_num" builder in

let i_ptr = L.build_malloc i32_t "ret_ptr" builder in
ignore( L.build_store i_int i_ptr builder );

let prev_node_next_ptr = L.build_malloc (pointer_t node_t) "prev_ptr"
builder in
let node_i = list_iter_node builder node_ptr i_ptr in

(* Determine if we are extracting zero'th element *)
let pred_expr = extract_num_val i builder in
let bool_val = L.build_fcmp L.Fcmp.Ole pred_expr f_zero "is_remove_zero"
```

```
      builder in
      let merge_bb = L.append_block context "remove_merge" the_function in

      let j_ptr = L.build_malloc i32_t "ret_ptr" builder in
      let then_bb = L.append_block context "then_remove_zero" the_function in
      let then_builder = L.builder_at_end context then_bb in
      (* get pointer to first element *)
      ignore( L.build_store (extract_list lst then_builder) prev_node_next_ptr
      then_builder );
      ignore( L.build_br merge_bb then_builder);

      let else_bb = L.append_block context "else" the_function in
      let else_builder = L.builder_at_end context else_bb in
      (* One to the left of one to be removed  *)
      let j = L.build_sub i_int one "sub" else_builder in
      ignore( L.build_store j j_ptr else_builder );
      let node_j = list_iter_node else_builder node_ptr2 j_ptr in
      let next_node_pointer_j = extract_node_node node_j else_builder in
      ignore( L.build_store next_node_pointer_j prev_node_next_ptr
      else_builder );
      ignore (L.build_br merge_bb else_builder);

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

      let next_node_pointer_i = extract_node_node node_i builder in
      let next_node_i = L.build_load next_node_pointer_i "next_node_i"
      builder in
      let prev_node_next = L.build_load prev_node_next_ptr "prev_node_next"
      builder in
      ignore( L.build_store next_node_i prev_node_next builder );
      in

      (* handle_plus operator returns the appropriate struct when applying '+' to
      two structs s1 and s2. handle_plus also takes the type of the structs as
      a i32 in type_structs to determine which functionality to apply
*)
      let handle_plus struct_type s1 s2 builder =
      let ret_ptr = L.build_malloc (pointer_t struct_t) "ret_ptr" builder in
      let merge_bb = L.append_block context "merge_equal" the_function in

      let plus_num_bb = L.append_block context "plus_num" the_function in
      let plus_str_bb = L.append_block context "plus_str" the_function in

      let num_builder = L.builder_at_end context plus_num_bb in
      let val1 = extract_num_val s1 num_builder and
             val2 = extract_num_val s2 num_builder in
      ignore( L.build_store ( construct_num (L.build_fadd val1 val2 "add"
             num_builder) num_builder ) ret_ptr num_builder);
      ignore (L.build_br merge_bb num_builder);
```

```
let str_builder = L.builder_at_end context plus_str_bb in
let str1 = extract_string s1 str_builder and
       str2 = extract_string s2 str_builder in
let char_ptr1 = L.build_in_bounds_gep str1 [| zero; zero |]
       "char_ptr" str_builder and
       char_ptr2 = L.build_in_bounds_gep str2 [| zero; zero |]
       "char_ptr" str_builder in
let len1 = L.build_call strlen_func [| char_ptr1 |] "len1"
       str_builder and
       len2 = L.build_call strlen_func [| char_ptr2 |] "len2"
       str_builder in
(* Allocate memory for new string of just the right size *)
let size = L.build_add (L.build_add len1 len2 "sum" str_builder) one
"sum" str_builder in
let ret_str = L.build_malloc string_t "str_cat_alloc" builder in
let ret_str_start = L.build_in_bounds_gep ret_str [| zero; zero |]
"ret_str_start" str_builder in
(* Move the first string to the return location *)
ignore( L.build_call strcpy_func [| ret_str_start; char_ptr1 |]
       "strcpy" str_builder );
(* Do the string concatenation *)
ignore( L.build_call strcat_func [| ret_str_start; char_ptr2 |]
       "strcat" str_builder );
(* Load the return string into the return pointer *)
ignore( L.build_store ( construct_str ret_str str_builder ) ret_ptr
       str_builder );
ignore (L.build_br merge_bb str_builder);

let switch = L.build_switch struct_type merge_bb 2 builder in
L.add_case switch zero plus_num_bb; (* Plus on two numbers *)
L.add_case switch one  plus_str_bb; (* Plus on two strings *)

L.position_at_end merge_bb builder;
L.build_load ret_ptr "data" builder
in

(* Assignment helper. d is an expression; s is a struct. *)
let rec assign d s builder =
(match d with
A.Id x ->
       (if (StringHash.mem local_vars x) then
       let new_val = L.build_load s "new_val" builder in
       let var = StringHash.find local_vars x in
       ignore (L.build_store new_val var builder)
       else
       let new_val = L.build_load s "new_val" builder in
       let var = L.build_malloc struct_t "var" malloc_builder in
       ignore (L.build_store new_val var builder);
       ignore (StringHash.add local_vars x var)); s
| _ ->
       let new_val = L.build_load s "new_val" builder in
       let d' = expr builder d in
       ignore (L.build_store new_val d' builder); s
)
(* Handle YAGL expressions *)
and expr builder = function
(* Construct an empty structure for NULL *)
A.Null -> build_empty_struct builder "NULL"
(* Given a number, create a struct containing that number *)
| A.Num i ->
let num_ptr = L.define_global (string_of_float i)
                                  (L.const_float double_t i)
                                  the_module in
let num_struct = build_empty_struct builder "num_struct" in
let struct_num = L.build_in_bounds_gep num_struct [| zero; zero |]
"struct_num" builder in
ignore (L.build_store num_ptr struct_num builder); num_struct
(* Given a string, create a struct containing that string *)
| A.String s ->
let str_ptr =
(L.const_bitcast (L.build_global_stringptr s s builder)
                    (pointer_t string_t)) in
let str_struct = build_empty_struct builder "str_struct" in
let struct_str = L.build_in_bounds_gep str_struct [| zero; one |]
"struct_str" builder in
ignore (L.build_store str_ptr struct_str builder); str_struct
(* Given a list, create a struct containing that list *)
| A.List l ->
let node = build_empty_node builder "node" in
let list_struct = build_empty_struct builder "list_struct" in
let struct_list = L.build_in_bounds_gep list_struct [| zero; two |]
"struct_list" builder in
ignore (L.build_store node struct_list builder);
(* Recursively construct the list *)
let rec make_list l node = (match l with
[] -> ();
| head :: tail ->
       let node_struct = L.build_in_bounds_gep node [| zero; one |]
       "node_struct" builder in
       let node_node = L.build_in_bounds_gep node [| zero; two |]
       "node_node" builder in
       let next_node = (build_empty_node builder "next_node") in
       ignore (L.build_store (expr builder head) node_struct builder);
       ignore (L.build_store next_node node_node builder);
       make_list tail next_node
)
in make_list l node;
list_struct
| A.Map m ->
```

```
let node = build_empty_node builder "node" in
let map_struct = build_empty_struct builder "map_struct" in
let struct_map = L.build_in_bounds_gep map_struct [| zero; three |]
"struct_map" builder in
ignore (L.build_store node struct_map builder);
(* Recursively construct the map *)
let rec make_map m node = (match m with
[] -> ();
| head :: tail ->
       let key = fst head in
       let value = snd head in
       let node_string = L.build_in_bounds_gep node [| zero; zero |]
       "node_string" builder in
       let node_struct = L.build_in_bounds_gep node [| zero; one |]
       "node_struct" builder in
       let node_node = L.build_in_bounds_gep node [| zero; two |]
       "node_node" builder in
       let next_node = (build_empty_node builder "next_node") in
       let str_ptr =
       (L.const_bitcast (L.build_global_stringptr key key builder)
                           (pointer_t string_t)) in
       ignore (L.build_store str_ptr node_string builder);
       ignore (L.build_store (expr builder value) node_struct builder);
       ignore (L.build_store next_node node_node builder);
       make_map tail next_node
)
in make_map m node;
map_struct
(* TODO: kill graph; rename digraph to graph *)
| A.Graph -> construct_graph builder
| A.Digraph -> construct_graph builder
(* Retrieve a variable, loading it from its pointer under the hood. *)
| A.Id s -> (try StringHash.find local_vars s
       with Not_found -> raise (Uninitialized_var(s)))
| A.Unop ( op, e1 ) ->  (* Handle unary operators *)

let s1 = expr builder e1 in
ignore( check_struct_type s1 num_type builder );
let e1' = extract_num_val s1 builder in
( match op with
A.Neg -> construct_num (L.build_fneg e1' "neg" builder) builder
| A.Not -> construct_num (L.build_uitofp (L.build_not (
       L.build_fcmp L.Fcmp.One e1' f_zero
       "neq" builder) "not" builder ) double_t "fp" builder ) builder
)
| A.Eqop( e1, op, e2 ) -> (* Handle comparison operators *)
let s1 = expr builder e1 and
       s2 = expr builder e2 in
( match op with
A.Equal -> is_equal s1 s2 builder
```

```
| A.Neq -> construct_num ( L.build_uitofp (
       L.build_not ( L.build_fcmp L.Fcmp.One (extract_num_val
       (is_equal s1 s2 builder) builder ) f_zero
       "neq" builder) "not" builder ) double_t "fp" builder ) builder
)
| A.Binop (e1, op, e2) ->  (* Handle binary operators *)
(* Stuct_types is an int indicating the types of the two expressions
       for use in overridden operator switch *)
let s1 = expr builder e1 and
       s2 = expr builder e2 in
(* Ensures the two expressions are of the same type *)
let struct_type = type_comparison s1 s2 builder in
(* Move num type checking down to lower level matches later as necessary *)
if op == A.Add then handle_plus struct_type s1 s2 builder
else ( ignore( check_index_type struct_type num_type builder );
let e1' = extract_num_val (expr builder e1) builder and
       e2' = extract_num_val (expr builder e2) builder in
(match op with
A.Sub -> construct_num (L.build_fsub e1' e2' "sub" builder) builder
| A.Mult -> construct_num (L.build_fmul e1' e2' "mult" builder) builder
| A.Div -> construct_num (L.build_fdiv e1' e2' "div" builder) builder
| A.Mod -> let unadjusted_mod = L.build_frem e1' e2' "mod" builder in
             let res_non_zero   = L.build_fcmp L.Fcmp.One unadjusted_mod
             f_zero "res_non_zero" builder in
             let lval_is_neg     = L.build_fcmp L.Fcmp.Olt e1' f_zero
             "negative" builder  in
             let adj_bool        = L.build_and lval_is_neg res_non_zero
             "and" builder in
             let f_bool          = L.build_uitofp adj_bool double_t "fp"
             builder in
             (* Only adjust by rval if result is non-zero and lval < 0 *)
             let adjust_factor = L.build_fmul f_bool e2' "scale"
             builder in
             let ret = L.build_fadd unadjusted_mod adjust_factor "adjust"
             builder in
             construct_num ret builder
| A.Less -> construct_num (L.build_uitofp (L.build_fcmp L.Fcmp.Olt
       e1' e2' "less" builder) double_t "fp" builder) builder
| A.Leq -> construct_num (L.build_uitofp (L.build_fcmp L.Fcmp.Ole
       e1' e2' "leq" builder) double_t "fp" builder) builder
| A.Greater -> construct_num (L.build_uitofp (L.build_fcmp L.Fcmp.Ogt
       e1' e2' "greater" builder) double_t "fp" builder) builder
| A.Geq -> construct_num (L.build_uitofp (L.build_fcmp L.Fcmp.Oge
       e1' e2' "geq" builder) double_t "fp" builder) builder
| A.And -> construct_num (L.build_uitofp (L.build_and
       (L.build_fcmp L.Fcmp.One e1' (L.const_float double_t 0.0) "neq"
       builder)  (* Map first and second *)
       (L.build_fcmp L.Fcmp.One e2' (L.const_float double_t 0.0) "neq"
       builder)  (* Expression to canonical *)
       (* Truth values then and *)
       "and" builder) double_t "fp" builder) builder
| A.Or -> construct_num (L.build_uitofp (L.build_or
       (L.build_fcmp L.Fcmp.One e1' (L.const_float double_t 0.0) "neq"
       builder)  (* Map first and second *)
       (L.build_fcmp L.Fcmp.One e2' (L.const_float double_t 0.0) "neq"
       builder)  (* Expression to canonical *)
       (* Truth values then or *)
       "or" builder) double_t "fp" builder) builder
) )
(* Assign a variable. If it is new, mallocte a pointer pointing to it. If it
exists and is being reassigned, redirect the pointer. *)
| A.Assign (d, e) -> assign d (expr builder e) builder
(* Access a list by an index *)
| A.ListOrMapAccess (d, e) ->
access_list_or_map builder (expr builder d) (expr builder e)
(* Access a map by string *)
| A.MapStringAccess (d, s) ->
let str_ptr =
(L.const_bitcast (L.build_global_stringptr s s builder)
                    (pointer_t string_t)) in
let map_struct = expr builder d in
ignore( check_struct_type map_struct map_type builder );
let map = extract_map map_struct builder in
access_map builder map str_ptr
(* Call print *)
(* Note print returns the struct it printed *)
| A.Call ("print", [e]) -> let e' = expr builder e in
print_struct builder e'; e'
| A.Call ("addV", [d; e]) ->
let g_struct = expr builder d and
       s_struct = expr builder e in
ignore( check_struct_type g_struct graph_type builder );
ignore( check_struct_type s_struct str_type builder );
let g = extract_graph g_struct builder and
       s = extract_string s_struct builder in
let vs = extract_graph_v g builder in
let v = access_map builder vs s in
(* TODO check if v's struct is already initialized *)
let v_attr_map_ptr = L.build_in_bounds_gep v [| zero; three |]
"v_attr_map_ptr" builder in
let v_attr_map = L.build_load v_attr_map_ptr "v_attr_map" builder in

let merge_bb = L.append_block context "merge" the_function in

let map_ptr = L.build_malloc (pointer_t node_t) "map_ptr" builder in

let then_bb = L.append_block context "then" the_function in
let then_builder = L.builder_at_end context then_bb in
let new_map = build_empty_node then_builder "map" in
ignore (L.build_store new_map map_ptr then_builder);
ignore (L.build_store new_map v_attr_map_ptr then_builder);
ignore (L.build_br merge_bb then_builder);

let else_bb = L.append_block context "else" the_function in
let else_builder = L.builder_at_end context else_bb in
let curr_map = extract_map v else_builder in
ignore (L.build_store curr_map map_ptr else_builder);
ignore (L.build_br merge_bb else_builder);

let is_null = L.build_is_null v_attr_map "is_null" builder in
ignore (L.build_cond_br is_null then_bb else_bb builder);
L.position_at_end merge_bb builder;

let map = L.build_load map_ptr "map" builder in

let label_str = extract_string (expr builder (A.String "label"))
builder in
let label_struct = access_map builder map label_str in
let str_ptr = L.build_in_bounds_gep label_struct [| zero; one |]
"str_ptr" builder in
ignore (L.build_store s str_ptr builder);
v
| A.Call ("v", [e]) ->
let g_struct = expr builder e in
let g = extract_graph g_struct builder in
let vs = extract_graph_v g builder in
ignore( check_struct_type g_struct graph_type builder );
let ret_struct = build_empty_struct builder "ret_struct" in
(* TODO this return structure doesn't seem right *)
let map_ptr = L.build_in_bounds_gep ret_struct [| zero; three |]
"map_ptr" builder in
ignore (L.build_store vs map_ptr builder);
ret_struct
| A.Call ("addE", [c; d; e]) ->
let g_struct = expr builder c and
       s_struct = handle_plus one
       (handle_plus one (expr builder d)
                           (expr builder (A.String "~")) builder)
       (expr builder e) builder and
       s1_struct = expr builder d and
       s2_struct = expr builder e in
ignore( check_struct_type g_struct graph_type builder );
ignore( check_struct_type s_struct str_type builder );
ignore (expr builder (A.Call ("addV", [c; d])));
ignore (expr builder (A.Call ("addV", [c; e])));
let g = extract_graph g_struct builder and
       s = extract_string s_struct builder and
       s1 = extract_string s1_struct builder and
       s2 = extract_string s2_struct builder in
let es = extract_graph_e g builder in
let e = access_map builder es s in

let e_attr_map_ptr = L.build_in_bounds_gep e [| zero; three |]
"e_attr_map_ptr" builder in
let e_attr_map = L.build_load e_attr_map_ptr "e_attr_map" builder in

let map_ptr = L.build_malloc (pointer_t node_t) "map_ptr" builder in

let merge_bb = L.append_block context "merge" the_function in

let then_bb = L.append_block context "then" the_function in
let then_builder = L.builder_at_end context then_bb in
let new_map = build_empty_node builder "new_map" in
ignore (L.build_store new_map e_attr_map_ptr then_builder);
ignore (L.build_store new_map map_ptr then_builder);
ignore (L.build_br merge_bb then_builder);

let else_bb = L.append_block context "else" the_function in
let else_builder = L.builder_at_end context else_bb in
let curr_map = extract_map e else_builder in
ignore (L.build_store curr_map map_ptr else_builder);
ignore (L.build_br merge_bb else_builder);

let is_null = L.build_is_null e_attr_map "is_null" builder in
ignore (L.build_cond_br is_null then_bb else_bb builder);
L.position_at_end merge_bb builder;

let map = L.build_load map_ptr "map" builder in

let vs = extract_graph_v g builder in
let v0_map = extract_map (access_map builder vs s1) builder in
let v0_str = extract_string (expr builder (A.String "orig")) builder in
let e_v0_struct = access_map builder map v0_str in
let e_v0_ptr = L.build_in_bounds_gep e_v0_struct [| zero; three |]
"e_v0_ptr" builder in
ignore (L.build_store v0_map e_v0_ptr builder);
let v1_map = extract_map (access_map builder vs s2) builder in
let v1_str = extract_string (expr builder (A.String "dest")) builder in
let e_v1_struct = access_map builder map v1_str in
let e_v1_ptr = L.build_in_bounds_gep e_v1_struct [| zero; three |]
"e_v1_ptr" builder in
ignore (L.build_store v1_map e_v1_ptr builder);
e
| A.Call ("e", [e]) ->
let g_struct = expr builder e in
ignore( check_struct_type g_struct graph_type builder );
(* TODO also take graphs (not just digraphs) into account *)
let g = extract_graph g_struct builder in
let es = extract_graph_e g builder in
let ret_struct = build_empty_struct builder "ret_struct" in
```

```
(* TODO this return structure doesn't seem right *)
let map_ptr = L.build_in_bounds_gep ret_struct [| zero; three |]
"map_ptr" builder in
ignore (L.build_store es map_ptr builder);
ret_struct
(* Call any other function *)
| A.Call ("append", [d; e]) ->
let l_struct = expr builder e in
let e_struct = expr builder d in
ignore( check_struct_type l_struct lst_type builder );
list_append l_struct e_struct builder; l_struct
(* Remove expects an index and a list struct *)
| A.Call ("remove", [i; ls]) ->
let ls' = expr builder ls and
      i' = expr builder i  in
let merge_bb = L.append_block context "merge_remove" the_function in

let ret_ptr = L.build_malloc (pointer_t struct_t) "ret_ptr" builder in

let bad_type_bb = L.append_block context "bad_type" the_function in
let bad_type_builder = L.builder_at_end context bad_type_bb in
ignore (L.build_call printf_func [| right_type_str |] "printf"
bad_type_builder);
ignore (L.build_call abort_func [||] "" bad_type_builder);
ignore (L.build_br merge_bb bad_type_builder);

let list_bb = L.append_block context "remove_list" the_function in
let map_bb = L.append_block context "remove_map" the_function in

let list_builder = L.builder_at_end context list_bb in
let ret_val = access_list_or_map list_builder ls' i' in
ignore( check_struct_type ls' lst_type list_builder );
ignore( remove_list i' ls' list_builder );
(*  let update_node = remove_list i' ls' in *)
(* now you set the nxt_node pointer on update node to be the nxt_node
      pointer on ret_val *)
ignore (L.build_store ret_val ret_ptr list_builder);
ignore (L.build_br merge_bb list_builder);

let map_builder = L.builder_at_end context map_bb in
let ret_val = access_list_or_map map_builder ls' i' in
ignore( check_struct_type ls' map_type map_builder );
ignore (remove_map i' ls' map_builder);
ignore (L.build_store ret_val ret_ptr map_builder);
ignore (L.build_br merge_bb map_builder);

let i_type = valid_struct_index i' builder in
let switch = L.build_switch i_type bad_type_bb 2 builder in
L.add_case switch zero list_bb;
L.add_case switch one map_bb;
```

```
L.position_at_end merge_bb builder;
L.build_load ret_ptr "ret_val" builder;
| A.Call (f, act) -> let (fdef, _) = StringMap.find f function_decls in
let actuals = List.map (expr builder) act in
let result = f ^ "_result" in
L.build_call fdef (Array.of_list actuals) result builder
| A.Typeof(e)->let e' = expr builder e in
             let struct_type = valid_struct_index e' builder in
             construct_num (L.build_sitofp struct_type double_t "fp"
             builder)  builder
| A.Noexpr -> build_empty_struct builder "empty_struct"
in

let rec build_for_kv_list k v e body builder =
let s = expr builder e in
let node_ptr = L.build_malloc (pointer_t node_t) "node_ptr"
builder in
let i_ptr = L.build_malloc double_t "i_ptr" builder in
ignore (L.build_store (extract_list s builder) node_ptr builder);
ignore (L.build_store f_zero i_ptr builder);

let pred_bb = L.append_block context "for_each_while" the_function in
ignore (L.build_br pred_bb builder);

let body_bb = L.append_block context "for_each_iter_body" the_function in
let body_builder = L.builder_at_end context body_bb in
let i = L.build_load i_ptr "i" body_builder in
let next_i = L.build_fadd i f_one "next_i" body_builder in
let node = L.build_load node_ptr "node" body_builder in
let next_node = extract_node_node node body_builder in
let node_struct = extract_node_struct node body_builder in
ignore (assign k (construct_num i body_builder) body_builder);
ignore (assign v node_struct body_builder);
ignore (L.build_store next_node node_ptr body_builder);
ignore (L.build_store next_i i_ptr body_builder);
let body_builder = (List.fold_left stmt body_builder body) in
add_terminal body_builder (L.build_br pred_bb);

let pred_builder = L.builder_at_end context pred_bb in

let node = L.build_load node_ptr "node_ptr" pred_builder in
let node_struct = extract_node_struct node pred_builder in
let bool_val = L.build_is_not_null node_struct "bool_val" pred_builder in

let merge_bb = L.append_block context "merge_for_each" the_function in
ignore (L.build_cond_br bool_val body_bb merge_bb pred_builder);
L.position_at_end merge_bb builder;

and build_for_kv_map k v e body builder =
let s = expr builder e in
let node_ptr = L.build_malloc (pointer_t node_t) "node_ptr"
builder in
ignore (L.build_store (extract_map s builder) node_ptr builder);

let pred_bb = L.append_block context "for_each_while" the_function in
ignore (L.build_br pred_bb builder);

let body_bb = L.append_block context "for_each_iter_body" the_function in
let body_builder = L.builder_at_end context body_bb in
let node = L.build_load node_ptr "node" body_builder in
let next_node = extract_node_node node body_builder in
let node_str = extract_node_string node body_builder in
let node_struct = extract_node_struct node body_builder in
let str_struct = construct_str node_str body_builder in
ignore (assign k str_struct body_builder);
ignore (assign v node_struct body_builder);
ignore (L.build_store next_node node_ptr body_builder);
let body_builder = (List.fold_left stmt body_builder body) in
add_terminal body_builder (L.build_br pred_bb);

let pred_builder = L.builder_at_end context pred_bb in

let node = L.build_load node_ptr "node_ptr" pred_builder in
let node_struct = extract_node_struct node pred_builder in
let bool_val = L.build_is_not_null node_struct "bool_val" pred_builder in

let merge_bb = L.append_block context "merge_for_each" the_function in
ignore (L.build_cond_br bool_val body_bb merge_bb pred_builder);
L.position_at_end merge_bb builder;

and build_for_each_list x e body builder =
let s = expr builder e in
let node_ptr = L.build_malloc (pointer_t node_t) "node_ptr"
builder in
let i_ptr = L.build_malloc double_t "i_ptr" builder in
ignore (L.build_store (extract_list s builder) node_ptr builder);
ignore (L.build_store f_zero i_ptr builder);

let pred_bb = L.append_block context "for_each_while" the_function in
ignore (L.build_br pred_bb builder);

let body_bb = L.append_block context "for_each_iter_body" the_function in
let body_builder = L.builder_at_end context body_bb in
let i = L.build_load i_ptr "i" body_builder in
let next_i = L.build_fadd i f_one "next_i" body_builder in
let node = L.build_load node_ptr "node" body_builder in
let next_node = extract_node_node node body_builder in
ignore (assign x (construct_num i body_builder) body_builder);
ignore (L.build_store next_node node_ptr body_builder);
ignore (L.build_store next_i i_ptr body_builder);
let body_builder = (List.fold_left stmt body_builder body) in
add_terminal body_builder (L.build_br pred_bb);

let pred_builder = L.builder_at_end context pred_bb in

let node = L.build_load node_ptr "node_ptr" pred_builder in
let node_struct = extract_node_struct node pred_builder in
let bool_val = L.build_is_not_null node_struct "bool_val" pred_builder in

let merge_bb = L.append_block context "merge_for_each" the_function in
ignore (L.build_cond_br bool_val body_bb merge_bb pred_builder);
L.position_at_end merge_bb builder;

and build_for_each_map x e body builder =
let s = expr builder e in
let node_ptr = L.build_malloc (pointer_t node_t) "node_ptr"
builder in
ignore (L.build_store (extract_map s builder) node_ptr builder);

let pred_bb = L.append_block context "for_each_while" the_function in
ignore (L.build_br pred_bb builder);

let body_bb = L.append_block context "for_each_iter_body" the_function in
let body_builder = L.builder_at_end context body_bb in
let node = L.build_load node_ptr "node" body_builder in
let next_node = extract_node_node node body_builder in
let node_str = extract_node_string node body_builder in
let str_struct = construct_str node_str body_builder in
ignore (assign x str_struct body_builder);
ignore (L.build_store next_node node_ptr body_builder);
let body_builder = (List.fold_left stmt body_builder body) in
add_terminal body_builder (L.build_br pred_bb);

let pred_builder = L.builder_at_end context pred_bb in

let node = L.build_load node_ptr "node_ptr" pred_builder in
let node_struct = extract_node_struct node pred_builder in
let bool_val = L.build_is_not_null node_struct "bool_val" pred_builder in

let merge_bb = L.append_block context "merge_for_each" the_function in
ignore (L.build_cond_br bool_val body_bb merge_bb pred_builder);
L.position_at_end merge_bb builder;

(* Handle YAGL statements *)
and stmt builder = function
(* Handle a simple expression *)
A.Expr e -> ignore (expr builder e); builder
(* Return void if this is the main function; otherwise, return the value
represented by the expression *)
```

```
(* TODO: add a semantic check that main doesn't return *)
| A.Return e -> ignore (match fdecl.A.fname with
"main" -> L.build_ret_void builder
| _ -> L.build_ret (expr builder e) builder); builder
(* If/else handling and block construction *)
| A.If (predicate, then_stmts, else_stmts) ->
let pred_struct = expr builder predicate in
ignore( check_struct_type pred_struct num_type builder );
let pred_expr = extract_num_val pred_struct builder in
let bool_val = is_true pred_struct builder in
let merge_bb = L.append_block context "merge" the_function in

let then_bb = L.append_block context "then" the_function in
add_terminal (
List.fold_left stmt (L.builder_at_end context then_bb) then_stmts)
(L.build_br merge_bb);

let else_bb = L.append_block context "else" the_function in
add_terminal (
List.fold_left stmt (L.builder_at_end context else_bb) else_stmts)
(L.build_br merge_bb);

ignore (L.build_cond_br bool_val then_bb else_bb builder);
L.builder_at_end context merge_bb
| A.While (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 (
List.fold_left 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 pred_struct = expr pred_builder predicate in
ignore( check_struct_type pred_struct num_type pred_builder );
let bool_val = is_true pred_struct pred_builder 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
| A.Foreach (x, e, body) -> let e' = expr builder e in
let merge_bb = L.append_block context "merge_for_each" the_function in

let bad_type_bb = L.append_block context "bad_type" the_function in
let bad_type_builder = L.builder_at_end context bad_type_bb in
ignore (L.build_call printf_func [| left_type_str |] "printf"
bad_type_builder);
ignore (L.build_call abort_func [||] "" bad_type_builder);
ignore (L.build_br merge_bb bad_type_builder);

let list_bb = L.append_block context "for_each_list" the_function in
let map_bb = L.append_block context "for_each_map" the_function in

let list_builder = L.builder_at_end context list_bb in
build_for_each_list x e body list_builder;
ignore (L.build_br merge_bb list_builder);

let map_builder = L.builder_at_end context map_bb in
build_for_each_map x e body map_builder;
ignore (L.build_br merge_bb map_builder);

let e_type = valid_struct_index e' builder in
let switch = L.build_switch e_type bad_type_bb 2 builder in
L.add_case switch two list_bb;
L.add_case switch three map_bb;

L.builder_at_end context merge_bb
| A.Forkv (k, v, e, body) -> let e' = expr builder e in
let merge_bb = L.append_block context "merge_for_kv" the_function in

let bad_type_bb = L.append_block context "bad_type" the_function in
let bad_type_builder = L.builder_at_end context bad_type_bb in
ignore (L.build_call printf_func [| left_type_str |] "printf"
bad_type_builder);
ignore (L.build_call abort_func [||] "" bad_type_builder);
ignore (L.build_br merge_bb bad_type_builder);

let list_bb = L.append_block context "for_kv_list" the_function in
let map_bb = L.append_block context "for_kv_map" the_function in

let list_builder = L.builder_at_end context list_bb in
build_for_kv_list k v e body list_builder;
ignore (L.build_br merge_bb list_builder);

let map_builder = L.builder_at_end context map_bb in
build_for_kv_map k v e body map_builder;
ignore (L.build_br merge_bb map_builder);

let e_type = valid_struct_index e' builder in
let switch = L.build_switch e_type bad_type_bb 2 builder in
L.add_case switch two list_bb;
L.add_case switch three map_bb;

L.builder_at_end context merge_bb
(* Do nothing. This happens if the statement is empty. *)
| A.Nothing -> builder
in
```

```
     (* Build the IR for the function's statements *)
     let builder = List.fold_left stmt builder fdecl.A.body in

     (* Point the variable memory malloction block to the statements block *)
     ignore (L.build_br stmts_bb malloc_builder);

     (* Return a null struct pointer from the function if we did not explicitly
     return before (except for the main function, which returns void) *)
     add_terminal builder (match fdecl.A.fname with
     "main" -> L.build_ret_void
     | _ -> L.build_ret (L.const_pointer_null (pointer_t struct_t)))
     in

List.iter build_function_body functions;
The_module
```

Ast.ml

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

type eqop = Equal | Neq

type op = Add | Sub | Mult | Div | Mod | Less | Leq | Greater |
       Geq | And | Or

type uop = Neg | Not

type expr =
       Num of float
   | Inf of float
   | True of float
   | False of float
   | Id of string
   | String of string
   | Eqop of expr * eqop * expr
   | Binop of expr * op * expr
   | Unop of uop * expr
   | Assign of expr * expr
   | Call of string * expr list
   | Typeof of expr
   | Null
   | Graph
   | Digraph
   | List of expr list
   | Set of expr list
   | Map of (string * expr) list
   | ListOrMapAccess of expr * expr
   | MapStringAccess of expr * string
   | Noexpr

type stmt =
   | Expr of expr
   | Return of expr
   | If of expr * (stmt list) * (stmt list)
   | Foreach of expr * expr * (stmt list)
   | Forkv of expr * expr * expr * (stmt list)
   | While of expr * (stmt list)
   | Break
   | Continue
   | Nothing

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

```
type program = func_decl list


(* Pretty-printing functions *)

let string_of_op = function
  Add -> "+"
| Sub -> "-"
| Mult -> "*"
| Div -> "/"
| Mod -> "%"
| Less -> "<"
| Leq -> "<="
| Greater -> ">"
| Geq -> ">="
| And -> "and"
| Or -> "or"

let string_of_eqop = function
Equal -> "=="
| Neq -> "!="
let string_of_uop = function
  Neg -> "-"
| Not -> "not "

let rec string_of_expr = function
  Num(n) -> string_of_float n
| Inf(_) -> "INF"
| True(_) -> "true"
| False(_) -> "false"
| Id(s) -> s
| String(s) -> s
| Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
| Unop(o, e) -> string_of_uop o ^ string_of_expr e
| Assign(v, e) -> string_of_expr v ^ " = " ^ string_of_expr e
| Call(f, l) -> f ^ "( " ^ String.concat ", " (List.map string_of_expr l) ^ " )"
| Typeof( e ) -> "typeOf" ^ "( " ^ string_of_expr e ^ " )"
| Null -> "NULL"
| Graph -> "graph"
| Digraph -> "digraph"
| List(l) -> "[ " ^ String.concat ", " (List.map string_of_expr l) ^ " ]"
| Set(s) -> "{ " ^ String.concat ", " (List.map string_of_expr s) ^ " }"
| Map(m) -> "{| " ^
      String.concat ", "
      (List.map (fun t -> fst t ^ " := " ^ string_of_expr (snd t)) m) ^
      " |}"
| ListOrMapAccess(l, i) -> string_of_expr l ^ "[" ^ string_of_expr i ^ "]"
| MapStringAccess(m, s) -> string_of_expr m ^ "." ^ s
| Noexpr -> ""
```

```
let rec string_of_stmt = function
  Expr(expr) -> string_of_expr expr ^ "\n"
| Return(expr) -> "return " ^ string_of_expr expr ^ "\n"
| If(e, sl, []) -> "if ( " ^ string_of_expr e ^ " )\n{\n" ^
       String.concat "" (List.map string_of_stmt sl) ^ "}\n"
| If(e, sl1, sl2) -> "if ( " ^ string_of_expr e ^ " )\n{\n" ^
       String.concat "" (List.map string_of_stmt sl1) ^ "}\nelse\n{\n" ^
       String.concat "" (List.map string_of_stmt sl2) ^ "}\n"
| Foreach(e1, e2, sl) -> "forEach ( " ^
       string_of_expr e1 ^ ", " ^ string_of_expr e2 ^ " )\n{\n" ^
       String.concat "" (List.map string_of_stmt sl) ^ "}\n"
| Forkv(e1, e2, e3, sl) -> "forComponentValue ( " ^
       string_of_expr e1 ^ ", " ^ string_of_expr e2 ^ string_of_expr e3 ^
       " )\n{\n" ^ String.concat "" (List.map string_of_stmt sl) ^ "}\n"
| While(e, sl) -> "forEach ( " ^ string_of_expr e ^ " )\n{\n" ^
       String.concat "" (List.map string_of_stmt sl) ^ "}\n"
| Break -> "break\n"
| Continue -> "continue\n"

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

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

```
Semant.ml
(* Semantic checking for the yagl compiler *)

open Ast
open String
module StringMap = Map.Make(String)

(* Semantic checking of a program. Returns void if successful,
   throws an exception if something is wrong.

   Check each global variable, then check each function *)

let check (functions) =

  (* Raise an exception if the given list has a duplicate *)
  let report_duplicate exceptf list =
      let rec helper = function
    n1 :: n2 :: _ when n1 = n2 -> (); raise (Failure (exceptf n1))
      | _ :: t -> helper t
      | [] -> ()
      in helper (List.sort compare list)
  in

(*   let check_empty execptf = function
      _ -> raise (Failure (execptf) )
  in *)

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

  (**** Checking Functions ****)

  if List.mem "print" (List.map (fun fd -> fd.fname) functions)
  then raise (Failure ("function print may not be defined")) else ();

  if List.mem "remove" (List.map (fun fd -> fd.fname) functions)
  then raise (Failure ("function remove may not be defined")) else ();

  if List.mem "append" (List.map (fun fd -> fd.fname) functions)
  then raise (Failure ("function remove may not be defined")) else ();

  if List.mem "v" (List.map (fun fd -> fd.fname) functions)
  then raise (Failure ("function remove may not be defined")) else ();

  if List.mem "e" (List.map (fun fd -> fd.fname) functions)
  then raise (Failure ("function remove may not be defined")) else ();
```

```
if List.mem "addV" (List.map (fun fd -> fd.fname) functions)
then raise (Failure ("function remove may not be defined")) else ();

if List.mem "addE" (List.map (fun fd -> fd.fname) functions)
then raise (Failure ("function remove may not be defined")) else ();

report_duplicate (fun n -> "duplicate function " ^ n)
     (List.map (fun fd -> fd.fname) functions);

(* Function declaration for a named function *)
let built_in_decls =
 (StringMap.add "v" { fname = "v"; formals = [("G")]; body = []}
 (StringMap.add "addV" { fname = "addV"; formals = [("G");("v")]; body = []}
 (StringMap.add "e" { fname = "e"; formals = [("G")]; body = []}
 (StringMap.add "addE" { fname = "addE"; formals = [("G");("v1");("v2")];
     body = []}
 (StringMap.add "print" { fname = "print"; formals = [("x")]; body = [] }
 (StringMap.add "remove" { fname = "remove"; formals = [("i");("ls")];
     body = [] }
 (StringMap.singleton "append" { fname = "append"; formals = [("l");("e")];
     body = [] }
 )))))))
in

let function_decls = List.fold_left (fun m fd -> StringMap.add fd.fname fd m)
                         built_in_decls functions
in

let function_decl s = try StringMap.find s function_decls
     with Not_found -> raise (Failure ("Unrecognized function " ^ s))
in

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

let check_function func =

     report_duplicate (fun n -> "duplicate formal " ^ n ^ " in " ^ func.fname)
     func.formals;

     (* Returns Boolean coresponding to if is valid lvalue in assignment
     Raises if invalid assignment *)
     let rec expr = function
     Num(_) -> false
     | True(_) -> false
     | False(_) -> false
     | Id(x) ->  true
     | String(_) -> false
     (* Binary/Equality operators are never valid lvalues *)
     | Binop(e1, _,e2) -> ignore( expr e1 ); ignore( expr e2 ); false
     | Eqop(e1, _,e2) -> ignore( expr e1 ); ignore( expr e2 ); false
     | Unop( _, e ) -> ignore( expr e ); false
     | Assign( e1, e2 ) -> let t1 = expr e1 in
     if not t1 then raise
            (Failure ("\n Illegal assignment!\n Can only assign to variables." ^
            " Cannot assign to litteral declarations, results of calls, or" ^
            " expressions\n Failed Expression: " ^
            string_of_expr e1 ^ "=" ^ string_of_expr e2 ^ "\n") )
     else expr e2
     | Call(fname, actuals) as call -> let fd = function_decl fname in
     if List.length actuals != List.length fd.formals then
            raise (Failure ("expecting " ^ string_of_int
            (List.length fd.formals) ^ " arguments in " ^ string_of_expr call))
     else
            ignore( List.map expr actuals );
            false;
     | Null -> false;
     | Graph -> false;
     | Digraph -> false;
     | List(exprs) -> ignore( List.map expr exprs ); false;
     | Set(exprs) -> ignore( List.map expr exprs ); false;
     | Map(pairs) -> ignore( report_duplicate (fun n -> "duplicate key " ^ n)
                         (List.map fst pairs) );
                   ignore( List.map expr (List.map snd pairs) );
                   false;
     | ListOrMapAccess(e1, e2) -> ignore (expr e1); ignore (expr e2); true;
     | MapStringAccess(_, _) -> true
     | Noexpr -> false
     | Inf(_) -> false
     | Typeof(_) -> false
     in

     (* Verify a statement or throw an exception *)
     let rec check_statement_list = function
            [Return _ as s] -> stmt s
     | Return _ :: _ -> raise (Failure "nothing may follow a return")
     | s :: ss -> stmt s ; check_statement_list ss
     | [] -> ()
     and

     stmt = function
     Expr e -> ignore( expr e )
     | Return e -> if func.fname = "main" then
                         raise (Failure ("You may not return in main function" ))
                   else ignore( expr e )
     | If( e, stmts1, stmts2 ) ->
            ignore (expr e);
            check_statement_list stmts1; check_statement_list stmts2;
     | Foreach(x, e, stmts ) -> ignore(expr e);
            (match x with
            Id(_) -> check_statement_list stmts;
```

```
               | _ -> raise (Failure  ("The first arg of ForEach must be an Id" ) )
                                    )
      | Forkv(x, y, e, stmts ) -> ignore (expr e);
               (match x,y with
               Id(_),Id(_) -> check_statement_list stmts;
               | _ -> raise
               (Failure("The first two args of ForComponentValue must be an Id" ) )
                                    )
      | While( e, stmts ) -> ignore (expr e); check_statement_list stmts;
      | Nothing -> ()
      in
      check_statement_list func.body
in

List.iter check_function functions
```

```
Runtests.sh
#!/bin/sh

# Regression testing script for yagl
# Step through a list of files
#  Compile, run, and check the output of each expected-to-work test
#  Compile and check the error of each expected-to-fail test

# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the yagl compiler.  Usually "./yagl.native"
# Try "_build/yagl.native" if ocamlbuild was unable to create a symbolic link.
YAGL="./yagl.native"
#YAGL="_build/yagl.native"

# Set time limit for all operations
ulimit -t 30

globallog=runtests.log
rm -f $globallog
error=0
globalerror=0

keep=0
delete=1
orp=''
Usage() {
	echo "Usage: runtests.sh [options] [.mc files]"
	echo "-k     Keep intermediate files"
	echo "-h     Print this help"
	echo "-d     Dont delete all intermediate files"
	exit 1
}

SignalError() {
	if [ $error -eq 0 ] ; then
    echo "FAILED"
    error=1
	fi
	echo "  $1"
}

# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile.  Differences, if any, written to difffile
Compare() {
	generatedfiles="$generatedfiles $3"
	echo diff -b $1 $2 ">" $3 1>&2
	diff -b "$1" "$2" > "$3" 2>&1 || {
    SignalError "$1 differs"
    echo "FAILED $1 differs from $2" 1>&2
       }
}

# Run <args>
# Report the command, run it, and report any errors
Run() {
       echo $* 1>&2 #redirects output to standard error
       eval $* || {
    SignalError "$1 failed on $*"
    return 1
       }
}

JustRun() {
       echo $* 1>&2 #redirects output to standard error
       eval $*
       return 0
}

# RunFail <args>
# Report the command, run it, and expect an error
RunFail() {
       echo $* 1>&2
       eval $* && {
    SignalError "failed: $* did not report an error"
    return 1
       }
       return 0
}

Check() {
       error=0
       basename=`echo $1 | sed 's/.*\\///
                            s/.y//'`
       reffile=`echo $1 | sed 's/.y$//'`
       basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

       echo -n "$basename..."

       echo 1>&2
       echo "###### Testing $basename" 1>&2

       generatedfiles=""

       generatedfiles="$generatedfiles ${basename}.ll ${basename}.out" &&
       JustRun "cat" "std.y" $1 "|" "$YAGL" ">" "${basename}.ll" &&
       JustRun "$LLI" "${basename}.ll" ">" "${basename}.out" &&
       Compare ${basename}.out ${reffile}.out ${basename}.diff
       # Report the status and clean up the generated files

       if [ $error -eq 0 ] ; then
    if [ $keep -eq 0 ] ; then
       rm -f $generatedfiles
    fi
    echo "OK"
    echo "###### SUCCESS" 1>&2
       else
    echo "###### FAILED" 1>&2
    globalerror=$error
       fi
}

CheckRun() {
       error=0
       basename=`echo $1 | sed 's/.*\\///
                    s/.y//'`
       reffile=`echo $1 | sed 's/.y$//'`
       basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

       echo -n "$basename..."

       echo 1>&2
       echo "###### Testing $basename" 1>&2

       generatedfiles=""

       generatedfiles="$generatedfiles ${basename}.ll ${basename}.out" &&
       JustRun "$YAGL" "<" $1 ">" "${basename}.ll" &&
       JustRun "$LLI" "${basename}.ll" ">" "${basename}.err" &&
       Compare ${basename}.err ${reffile}.err ${basename}.diff

       # Report the status and clean up the generated files

       if [ $error -eq 0 ] ; then
    if [ $keep -eq 0 ] ; then
       rm -f $generatedfiles
    fi
    echo "OK"
    echo "###### SUCCESS" 1>&2
       else
    echo "###### FAILED" 1>&2
    globalerror=$error
       fi
}

CheckFail() {
       error=0
       basename=`echo $1 | sed 's/.*\\///
                           s/.y//'`
       reffile=`echo $1 | sed 's/.y$//'`
       basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

       echo -n "$basename..."

       echo 1>&2
       echo "###### Testing $basename" 1>&2

       generatedfiles=""

       generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&
       JustRun "$YAGL" "<" $1 "2>" "${basename}.err" ">>" $globallog &&
       Compare ${basename}.err ${reffile}.err ${basename}.diff

       # Report the status and clean up the generated files

       if [ $error -eq 0 ] ; then
    if [ $keep -eq 0 ] ; then
       rm -f $generatedfiles
    fi
    echo "OK"
    echo "###### SUCCESS" 1>&2
       else
    echo "###### FAILED" 1>&2
    globalerror=$error
       fi
}

while getopts kdpsh c; do
       case $c in
    k) # Keep intermediate files
       keep=1
       ;;
    h) # Help
       Usage
       ;;
  d) #Delete intermediate files
       delete=0
       ;;
       esac
done

shift `expr $OPTIND - 1`

LLIFail() {
  echo "Could not find the LLVM interpreter \"$LLI\"."
  echo "Check LLVM installation and/or modify the LLI variable in runtests.sh"
  exit 1
```

```
}

which "$LLI" >> $globallog || LLIFail

if [ $# -ge 1 ]
then
       files=$@
else
       files="tests/fail-*.y tests/run-*.y tests/test-*.y"
fi

#Get the current OCAMLRUNPARAM for restoration
orp=`$OCAMLRUNPARAM`
eval "export OCAMLRUNPARAM=''" #Set to '' so that outputs on failure match

for file in $files
do
       case $file in
    *test-*)
       Check $file 2>> $globallog
       ;;
    *fail-*)
       CheckFail $file 2>> $globallog
       ;;
    *run-*)
       CheckRun $file 2>> $globallog
       ;;
    *)
       echo "unknown file type $file"
       globalerror=1
       ;;
       esac
done
#Restore OCAMLRUNPARAM
eval "export OCAMLRUNPARAM=" $orp

if [ $delete -eq 1 ] ; then
  eval  "rm *.diff *.ll* *.out *.log *.err"
fi

exit $globalerror
```