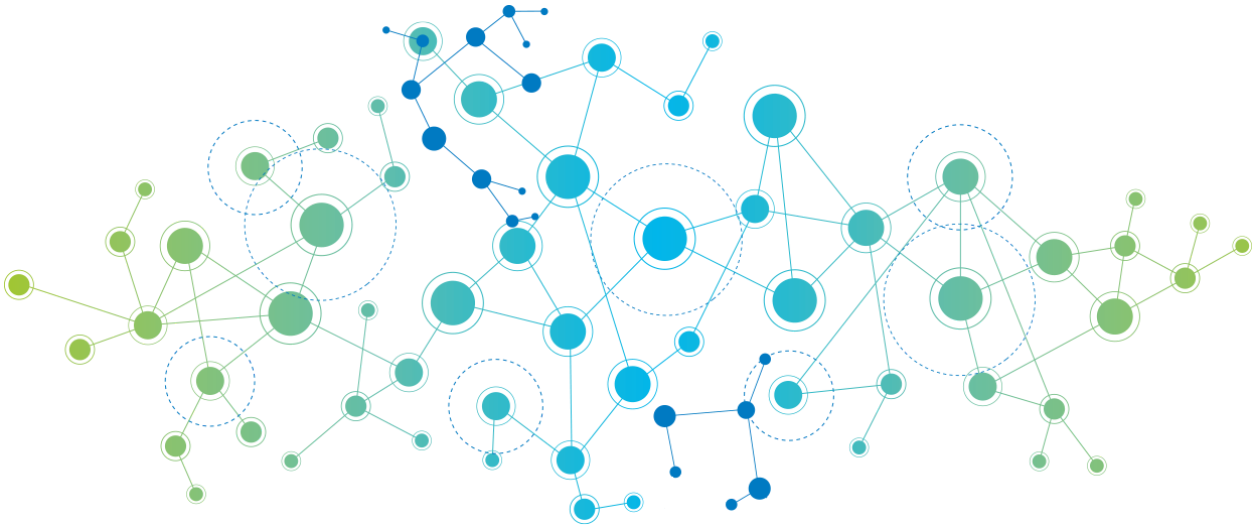


YAGL

Yet Another Graph Language

Final Report

Adam Carpentieri | AC4409
Jack Hurley | JTH2165
James Mastran | JAM2454
Shvetank Prakash | SP3816

1 Introduction

1.1 Background

YAGL is a programming language created for the purpose of building, augmenting, exploring, and analyzing graph-like data structures. Across various sub-disciplines in computer science, the graph data structure is pervasive and powerful which makes it a great candidate to be added to the list of classical data types. YAGL provides the building blocks, and syntax to make working with such data a figurative breeze. The language adopts a C-like syntax in order to facilitate a minimal learning curve.

Graphs are fundamental in data structures and algorithms. They are ubiquitous and can be used to represent almost anything: social media connections, roads that connect cities, flights between cities, relationships or friendships, and many other mathematical & logical problems. Using common graph operations & operators as our building blocks, YAGL includes a standard library of functions that has easily implemented some of the many of the widely used graph algorithms.

1.2 Related Work

Clearly, YAGL is not the first such attempt at simplifying the manipulation of graph data structures. Graph data structures were being implemented in less recent languages such as Algol at least as early as 1980[1]. Although we are not computer science historians, it seems likely, given Edsger W. Dijkstra's involvement with Algol earlier than this, and his famed discoveries with graphs; that programming languages and graph algorithms were developed in parallel.

With the development of object oriented languages, complex and abstract data structures were able to be represented with far less cognitive strain. One such recent example is the NetworkX library for Python[2].

Despite this rich palette already in existence, we boldly explore these charted waters, putting our own unique spin on the topic. "Yet another", perhaps, but one-of-a-kind nonetheless.

1.3 Goals

1.3.0.1 Familiarity

We chose the C syntax as a model for our language. It is the language most of us have been using for a very long time. The C syntax, for better or for worse, has stood the test of time. It does

[1] https://academic.oup.com/comjnl/article/23/3/237/375193?login=true
[2] https://networkx.org/

not have a ton of fancy syntactic sugar, it is very straightforward, and very precise. These are traits we value in both our programming languages and the people we work with.

Of course, with the addition of the graphing functionality, our team hemmed and hawed, so to speak, about the various ways we could represent certain actions. We feel that the final result of many iterations shown below is a **pretty good** representation of what we initially had in mind. It is a balance of familiarity and efficiency. There were constant objections about something not seeming “idiomatic”, and largely those objections were well founded.

1.3.0.2 Flexibility

Flexibility was at the forefront of our efforts. For instance, having an implicit main() function, and allowing function declarations in any order, at any point in the code allows for less rigid code.

There are typically several ways to accomplish the same thing in our language, depending on the context or preference of the user. In particular, the chaining of edge operations, and the instantiation of a bidirectional edge with a single arrow streamlines coding efforts.

1.3.0.3 Safety

By not exposing the entire C standard library and its ability to allocate memory on the heap, as well as removing pointer dereferencing and manipulation, we save the user from some potential pitfalls. It is a deliberate design choice to keep these features out of our language, as it is not simply C with some additions.

1.4 Notable Features of YAGL

Without explaining in too much detail, which is what the rest of this document does, here are the most notable features of YAGL:

- Arrays
- Implicit `main()` function
- Graph and Node types
- (Under-the-hood-recursive) operations to augment graphs in a smooth manner
- Preprocessing/Import capabilities
- Accessor functionality to get information about Graphs, Nodes, and Edges in Graphs.
- Generic printing capabilities
- C-like scoping principles
- Out-of-order function calling & declaration
- Many standard library functions implemented in YAGL

The rest of the document explains these in greater detail as well as the other minor features added.

2 Tutorial

2.1 Using the Compiler

Inside the base directory for YAGL, and in the docker container, type **make all**. This creates the yagl.native compiler. This, in turn, will convert YAGL code to LLVM IR that we then compile down to assembly and link with our C built-in functions using the C compiler (cc) ultimately to generate an executable.

There is a convenient shell script to compile YAGL code into an executable like this, as well as report any syntax errors. Inside the base project directory, simply type:

./yagl <name of yagl file *without* .ygl extension>.

To get a more detailed output about each stage of the compiler on a single input file, use:

./yagl_debugger <name of yagl file *without* .ygl extension>

Here is a sample hello world program which is compatible with YAGL syntax:

```
helloWorld();

int helloWorld() {
     String hw = "Hello World";
     print(hw);
}
```

***Please note that the above YAGL code block may include incomplete or inaccurate code highlighting.**

Compiling with the above mentioned instructions will produce an executable file with the same name as the file in which the YAGL code resides. Running the executable will output the expected “Hello World” text.

With this code, we are demonstrating the ability to declare a function after being first called. More on that in section 2.4.

2.2 Data Manipulation

2.2.0.1 Primitives

Declaring variables is a familiar task for anyone with even a small exposure to a C-like syntax.

- A newly declared variable is preceded by its type, such as **int, float, bool,** or **char.**

- This can be combined with an assignment operation by using the = symbol after the declaration followed by a valid value;

```
/* declaration followed by assignment */

int a;
a = 1;

/* or in one line */

float b = 2.1;

bool c = b < 2.0;
```

*Please note that the above YAGL code block may include incomplete or inaccurate code highlighting.

2.2.0.2 Derived Types

A similar convention follows our derived types such as **Arrays**, **String, Graph,** and **Node.** An important difference is in the assignment aspect: use of an = symbol is not used.

Note: **Edge** type cannot be declared on its own, but rather as part of a graph-related operation. This operation is explained in detail in the Language reference manual.

It is worth pointing out that these derived types have assignment values that can vary greatly in format to the derived types. Here is one such valid example:

```
Graph g;
Node a("Test");
```

*Please note that the above YAGL code block may include incomplete or inaccurate code highlighting.

2.2.0.3 Arrays

Arrays are indexed collections of values of a certain type. They can take in both primitive and derived types as indices.

The length of the array must be indicated inside square brackets at time of declaration. The expression inside the brackets can be any expression, as long as it evaluates to a positive integer.

As expected, arrays can be accessed by referencing the variable name assigned to the array, followed by square brackets which inside refer to a valid integer int the range of the array's index size.

When passing arrays as an argument to a function, it is passed by value. The entire array gets copied. This is a convention breaking decision, but we preferred it in the context of the library functions that were being built with our language.

```
int bar;
bar = 9;

int[10] foo;
foo[0] = 0;
foo[2+3] = 123;
foo[bar] = 456;

printInt(foo[3-3]);
printInt(foo[5]);
printInt(foo[bar]);

int temp;
temp = foo[0];
printInt(temp);
```

*Please note that the above YAGL code block may include incomplete or inaccurate code highlighting.
Above, this code is actually one of the automated tests our testing suite runs to ensure arrays work as intended.

2.3 Control Flow

YAGL control flow statements resemble C control flow statements, with the exception of the removal of the for loop. The while loop can be used to express the same semantic intent.

```
int a = 0

/* if-else */

if (a < 10) {
     print("a is less than 10");
} else {
     print("a is greater than or equal to 10");
}

/* only if */
if (a == 10) {
     print("a equals 10");
}
```

```
/* while loop */
while (a < 10) {
     print(a);
     a = a + 1;
}
```

*Please note that the above YAGL code block may include incomplete or inaccurate code highlighting.

2.4 Implicit Main Function & Out of Order Function Declarations

One of the cornerstone features of YAGL is the ability to write code without a main() function. This is more akin to a scripting language. Truth be told, we never intended for this to be an actual feature, until the professor exclaimed “cool” (or something to that effect) during one of our early demos. The demo code accidentally was missing the main() function. We embraced this idea at first to simply please the professor, but in retrospect we have enjoyed the flexibility it enables.

It was accomplished by stripping out all function decls in a first pass, then wrapping the remaining code in an implicit main() function. This then gets parsed as a valid C-like program.

As a result of this two-pass parsing operation, we gain the ability to declare functions in any order, and call them before they are declared, as an added bonus. The code below illustrates these features.

```
started();

void finished() {
      printString("Executed" + " Completely");
      printString("******************");
}

String[3] foo;
String[3] t;

t[0] = "Make";
t[1] = "Some";
t[2] = "Noise";

int bar = 0;

while(bar < 3) {
      foo[bar] = t[bar];
      printString(foo[bar]);
      bar = bar + 1;
```

```
}

finished();

void started() {
      printString("******************");
      printString("Loading" + " ...");
}
```

*Please note that the above YAGL code block may include incomplete or inaccurate code highlighting.

Language Reference Manual

3 (Original) Introduction

YAGL may be just that, Yet Another Graph Language, but it is unlike any other— hopefully. The pervasiveness of graphs in computer science makes them a great candidate to be added to the list of classical types that are widely used in other languages. This language aims to make implementing graphs and their algorithms much simpler and easier! While we are creating our own language syntax and design, we do plan on adopting some C's syntax & features that we appreciate most.

Graphs are fundamental in data structures and algorithms. They are ubiquitous and can be used to represent almost anything: social media connections, roads that connect cities, flights between cities, relationships or friendships, and many other mathematical & logical problems. Our language aims to simplify the use of graphs in computation by nicely wrapping many of the operations used in well known algorithms into a neat & compact syntax. Using these commonly used graph operations & operators as our building blocks, we have also built a Standard Library that has easily implemented many of the widely used graph algorithms.

4 Lexical Conventions

In YAGL, identifiers, keywords, constants, operators, and separators are all considered tokens. The lexeme associated with each token is composed of characters from the ASCII character set. Tokens must be separated using whitespace, comments, or any other kind of separator token.

4.1 Identifiers

Identifiers are used for naming and must begin with an alphabetical letter. This first character can then be followed by any sequence of digits, letters, and underscores. Names can be of any

length but must be unique. Uppercase and lowercase letters are distinct. The regular expression for identifiers is:

```
['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*
```

4.2 Keywords

The following words are reserved in YAGL for programming constructs, defining types, and special constants and may not be used in any other context such as naming:

```
while       int
if          char
else        bool
return      Graph
true        String
false       Node
void        float
import
```

Each lexeme associated with each of these tokens above is the keyword's respective spelling.

4.3 Constants

There are 5 different types constants used to represent literals:

4.3.0.1 Integer Constants

Integer constants are composed of any sequence of digits (0-9) and an optional hyphen (-) for the sign of the integer to represent a literal between -2,147,483,648 to 2,147,483,647. All integer constants are interpreted as base 10 numbers. The regular expression for integer constants is:

```
['0' - '9']+
```

4.3.0.2 Character Constants

Character constants are represented using single quotes around a single character from the ASCII character set encoding. Sometimes two characters are needed in special cases to represent the character. In these cases the first character is a backslash which is also called the escape character in this context:

‘\n’ : new line

‘\r’ : carriage return

‘\t’ : tab

‘\b’ : backspace

The regular expression for these escape characters is:

```
"'" ('\\' ['b' 't' 'r' 'n']) "'"
```

The regular expression for all other ASCII characters is:

```
"'" [' ' - '~'] "'"
```

4.3.0.3 Floating Point Constants

Floating point constants are composed of an integer sequence followed by a decimal point and then a fractional sequence. The integer sequence can contain an optional hyphen to show the sign followed by any sequence of digits to represent an integer literal between 1.2E-38 to 3.4E+38. The decimal point is a simple period (.) . The fractional portion can be any sequence of digits to represent six decimal points of precision. The first two components (integer portion, decimal point) are required for the constant to be interpreted as a floating point. The regular expression for floating point constants is:

```
['0' - '9']+ '.'  ['0' - '9']* ( ['e' 'E'] ['+' '-']? ['0' - '9'] )?
```

4.3.0.4 String Constants

String constants are represented using double quotes “” around a string literal, which is simply a sequence of ASCII characters (including escape chars). The regular expression for string constants is:

```
'"' (ascii* escapeChars*)+ '"'
```

where the regular expressions for `ascii` and `escapeChars` are defined in 2.3.0.2 above.

4.3.0.5 Boolean Constants

There are only two kinds of boolean constants and these literals are reserved as keywords in the language: `true and false`. The lexeme associated with each of these two tokens is the keyword’s respective spelling.

4.4 Operators

The following characters are reserved as operators:

```
+       -       /
*       =       !
<       >       &&
||      ==
```

Each one of these is considered a separate token with the exception of `-` followed by an integer or floating point constant.

4.5 Separators

There are 9 separator tokens in YAGL:

```
(       )
[       ]
{       }
.
;
,
```

These are used in the language to separate code in various ways and each have a different use case outlined later in the LRM. A left parenthesis, square bracket, or curly brace needs to be followed eventually by a respective right closing character.

4.6 Whitespace

All white space (spaces, tabs, new lines) are ignored and not considered tokens. They are simply used to separate other tokens.

4.7 Comments

All comments begin with a `/*` and end with `*/`. Comments are ignored and not considered tokens **but can be used to separate tokens**.

5 Data Types

Data types are split into two different categories: primitive and derived. Primitives are the base data types and consist of `int`, `char`, `bool`, `float` and `void`. Derived data types are built from

primitive data types. These types include `Node`, `Edge`, `Graph`, `Array` and `String`. **Note that bitwise representations of data types are not exposed to the user.**

5.1 Primitive

5.1.0.1 int

Used to store whole number values with a storage size of 32 bits. Integer types are stored in 2's complement and so the range is from -2,147,483,648 to 2,147,483,647. Characters (declared, and hereinafter called, char) are chosen from the ASCII set; they occupy the rightmost seven bits of an 8-bit byte.

5.1.0.2 char

Stores character values with a size of 8 bits. Character types are 1 byte unsigned integers and so their value range is from 0 to 255.

5.1.0.3 bool

Either stores the value `true` or `false`.

5.1.0.4 float

Stores floating point numbers (i.e. fractions/decimals). Floating types have a storage size of 32 bits and are formatted in IEEE 754 (1-bit for the sign, 8-bits for the exponent, 23-bits for the value). The value range is from 1.2E-38 to 3.4E+38 and has 6 decimal places of precision.

5.1.0.5 void

Specifies no value is available. It is used for functions to return nothing, functions to not take arguments and used to point to an address of an object but not its type.

5.2 Derived

5.2.0.1 Node

Contains one or more attributes of any type (string, int, bool, etc.). Similar to a dictionary in Python.

Adding a Node to the Graph is shown in the example section.

5.2.0.2 Edge

Edges are an implicit type. They exist only in Graphs. Edges cannot be declared like other data types in YAGL.

Connects two nodes and contains a reference to its source and destination nodes. It will hold an int which can correspond to the edge's weight. All edges are directed, but you can "create" a bidirectional edge by having two edges connecting the same nodes but in opposite directions.

The information stored in an Edge is the source node, destination node, and some associated descriptor which is of type int. All these are accessible. For example, to access the destination node, source node, or attribute (weight) of an Edge E, it is simply E.dest, E.src, or E.attr, respectively;

Adding an Edge to the Graph is shown in the example section.

5.2.0.3 Graph

A Graph stores two sets of arrays: an array for the nodes contained within the Graph and an array of Edges for relationships between the nodes. To keep track of the size of the arrays, two ints are stored as well. These arrays are pointers.

Graphs are mutable. Thus when a new Graph is created it does not contain any nodes nor any edges and therefore the edges and nodes arrays point to null. When a node is added into a Graph, if the Graph has room, the node is simply added. If the Graph does not have enough room, a copy is made that has a capability of holding more nodes, and the node to-be-added is added.

One can access the array of `edges` or `nodes` of a graph. For example, to access Graph G's nodes, one can do `G.node[0];`

Adding nodes and edges to a Graph is shown in the example section.

5.2.0.4 String

An array of characters terminated with a null character. String types contain a pointer to the array with a size of 8 bytes.

5.2.0.5 Array

A contiguous chunk of memory storing multiple instances of the same derived or primitive type. All arrays are of fixed length and one-dimensional.

6 Expressions and Operators

6.1 Unary

The unary operators are !, -

6.1.0.1 Accessor: variable.variable

The . operator is used on a variable of a specific type and accesses its internal data/variables.

6.1.0.2 Negation: !bool

The ! operator placed immediately before an expression is the negation operator. It works on `bool` types. If the `bool` is true, then the negation of the `bool` becomes false. If the `bool` is false, then the negation of the `bool` becomes true.

6.1.0.3 Negation: -expression

The - operator placed immediately before an expression is the negative operator. The result is the negative of the expression and works on `int` and `float` types.

6.1.0.4 Array accessor: []

The [index] operator is used after an array variable to access and dereference the contents of the memory location held in the index number (0 based) of the array.

6.1.0.5 expressions not supported

The ++ (increment) and -- (decrement) unary operators are not supported in our language since these are equivalent to `expression = expression + 1` or `expression = expression - 1`, respectively.

Bitwise operators are not supported at this time.

6.2 Binary

The binary operators are +, -, *, /, ==, <, >, =, &&, and ||.

6.2.0.1 Multiplication: expression * expression

The * operator with two expressions indicates multiplication. Both expressions must be of the same type. Therefore, this operator works on `int * int`, `float * float`, or `char * char` expressions. No other combinations are allowed. The result is another expression of the same type as the expressions used with this operator. This operator is left evaluated (grouped left-to-right).

6.2.0.2 Division: expression / expression

The / operator with two expressions indicates division. Both expressions must be of the same type. Therefore, this operator works on `int / int`, `float / float`, or `char / char` expressions. No other combinations are allowed. The result is another expression of the same

type as the expressions used with this operator. This operator is left evaluated (grouped left-to-right).

6.2.0.3 Addition: expression + expression

The + operator with two expressions indicates addition. Both expressions must be of the same type. Therefore, this operator works on `int + int`, `float + float`, or `char + char` expressions. No other combinations are allowed. The result is another expression of the same type as the expressions used with this operator. This operator is left evaluated (grouped left-to-right).

6.2.0.4 Subtraction: expression - expression

The - operator with two expressions indicates subtraction. Both expressions must be of the same type. Therefore, this operator works on `int - int`, `float - float`, or `char - char` expressions. No other combinations are allowed. The result is another expression of the same type as the expressions used with this operator. This operator is left evaluated (grouped left-to-right).

6.2.0.5 Equality: expression == expression

The == operator is the equal-to operation. It determines whether two expressions are equivalent of the form expression == expression and returns a bool type. One aspect to note is the lower precedence; for example, `a>b==c>d` is 1 if a > b and c > d or a < b and c < d (thus both have the same truth value). For comparing Nodes, Edges, or Graphs, it compares memory location. This operator is left-to-right evaluated.

6.2.0.6 Graph special operator: expression : expression

This operator is to define an expression to affect the graph on the left-hand side. The first expression must be a Graph and the second must be a valid operation on the Graph (such as the arrow operator or question mark operator). This operator is left-to-right evaluated.

6.2.0.7 Less-than: expression < expression

The < operator is the less-than operation. It determines whether the left side of the operator is less-than the right side and if so it yields true; otherwise, false. Both expressions must be of the same type. Does not apply to the bool type. This operator is left evaluated (grouped left-to-right).

6.2.0.8 Greater-than: expression > expression

The > operator is the greater-than operation. It determines whether the left side of the operator is greater-than the right side and if so it yields true; otherwise, false. Both expressions must be of the same type. Does not apply to the bool type. This operator is left evaluated (grouped left-to-right).

6.2.0.9 Arrow-operator: expression ->expression expression

The -> operator is the arrow operation. It is used to create an edge from the left-hand side directed towards the right-hand side expression. The expression in the middle is a literal (integer) number and is optional (defaults to 1). The middle expression must be an integer and the other two expressions must be a Node. This operator is left evaluated (grouped left-to-right).

6.2.0.10 Assignment: expression = expression

The assignment operator groups right-to left, that is the right side of the binary operator is assigned to the left side. The assignment operator returns a value that is equal to the right side of the assignment and is of the same type. Both expressions must be of the same type, no other combinations are allowed.

6.2.0.11 OR: bool || bool

The || operator yields true if either (or both) of the booleans are true; otherwise, it yields false. This operator guarantees left-to-right evaluation.

6.2.0.12 AND: bool && bool

The && operator yields true if (and only if) both of the booleans are true; otherwise, it yields false. This operator guarantees left-to-right evaluation.

6.2.0.13 expressions not supported

Greater than or equal and less than or equal comparators.

6.2.0.14 boolean short circuiting

Boolean expressions are **not** short circuited.

6.3 Operators Precedence

The precedence order is unary operators, multiplication and division, addition and subtraction, comparators (<, >, ==), the and/or logical operators, and then lastly the assignment operator.

Parenthesis in logical evaluation or arithmetic can be used to override inherent precedence. For example 2 + 3 * 4 should evaluate to 14, but (2+3) * 4 should evaluate to 20.

The overarching precedence is in order of the major sections above (i.e. unary operations receive higher precedence over binary operations).

To summarize precedence (in order of increasing precedence):

Operator Name	Symbol	Associativity
Assignment	=	Right-to-Left Associativity
Or	||	Left-to-Right Associativity
And	&&	Left-to-Right Associativity
Equal Graph-operator	== :	Left-to-Right Associativity
Less-than Greater-than	< >	Left-to-Right Associativity
Arrow-operator	->{optional: int}	Left-to-Right Associativity
Reverse-Arrow-operator	<-{int}	Left-to-Right Associativity
Bidirectional-Arrow-operator	{optional: int}<->{optional: int}	Left-to-Right Associativity
Plus Minus	+ -	Left-to-Right Associativity
Times Divide	* /	Left-to-Right Associativity
Not	!	Right-to-Left Associativity
Array-access-operator	array[expr]	Left-to-Right Associativity
Formals Separator	,	Left-to-right Associativity
Accessor-operator	.	Left-to-Right Associativity

7 Functions

7.1 Functions

Functions in our language will follow the syntax:

```
return_type function_name (type arg1, type arg2, …, type argN){
      /* function body */
}
```

The `return` statement is a mechanism for returning a value to the caller. Any expression can follow return:

```
      Return expression;
```

Our language does not support functions as pointers.

7.2 Calling a function

Once a function is defined, it may be called in any place inside another function. The syntax is

```
function_name(arg1, arg2, …, arg n);
```

8 Statements

8.1 Declarations

Declaration statements may be placed anywhere in the file or anywhere in a function. The following sections explain each declaration type in detail.

8.1.0.1 Primitive Data Types

To declare a primitive data type, the syntax is `type var_name = expression of same type;`

Exceptions: Nodes and Graphs. To declare a new Node, the syntax is `Node node_name("description");` and to declare a new Graph, the syntax is `Graph graph_name;` Both of these handle memory allocation.

8.1.0.2 Arrays

To declare an array, the syntax is `type[num_of_elements] name;`

8.1.0.3 Nodes

To declare a Node, the syntax is `Node node_name("description");`

8.1.0.4 Edges

An Edge is not declared directly. It is declared by adding an edge to a graph. Assume a Graph G exists contain Nodes A, B and C, two edges from A to B (with weight 5) and B to C (with weight 7) can be declared as:

```
G: A ->5 B ->7 C;
```

Some more examples include, in increasing level of difficulty:

```
G: A 5<- B ->7 C;
```

```
G: A -> B, A -> C;
```

```
G: A  5<->7 B 8<->9 C;

G: A  x<->x B;              /* x is an int */

G: A ->|x+1| B;             /* x is an int */

G: A [|x+1|<->|x*2|] B;     /* x is an int */
```

The bars (| |) are always optional unless expr is compounded (e.g. x + 1). The brackets ([ ]) around arrows are always optional except for one special case: `G: A [5<-] B;` where the user wants to put the expression on the left-hand side of the Reverse-Arrow.

8.1.0.5 Graph

To declare a Graph, the syntax is `Graph name;` which will create a graph type and store its address in the variable `name`.

8.1.0.6 String

To declare a string, the syntax is `String name = "contents of the string";` Since Strings are fixed arrays, their length cannot be changed.

8.2 Statements

Statements are executed in their order written in the code.

8.2.0.1 Expression Statement

Any expression of the form `expression;` is a statement.

8.2.0.2 Graph Operations Summary (a prelude)

As a summary, the following operations are valid on graph augmentations, assuming graph G, and nodes A, B, C:

G: + A + B;	Add nodes A and B to graph G.
G: A -> B;	Add edge with weight 1 from A to B.
G: A ->x B; **OR** G: A ->|x| B;	Add edge with weight of x. | | are optional.
G: A -> |x * 2| B;	Add edge with weight of x * 2. | | not optional.
G: A <- B;	Add edge with weight 1 from B to A.
G: A x<- B; **OR** G: A <- x B;	Add edge with weight x from B to A.
G: A [|x+1|<-] B; **OR** G: A <-|x+1| B;	Add edge with weight x+1 from B to A. [ ] are optional if first recursive call, otherwise required in

	the former case. | | are required. Note: Weight side is optional.
G: A <-> B;	Add edges with weights 1 from A to B and B to A.
G: A x<->y B; **OR** G: A [x<->y] B; **OR** G: A [|x|<->|y|] B;	Add edges with weights x and y, respectively from B to A and A to B. Both [ ] and | | are optional.
G: A [|x*2|<->|y+1|] B;	Add edges with weights x*2 and y+1, respectively from B to A and A to B. | | are required. [ ] are optional if first recursive call, otherwise required.
[ ]	Wrap the edge and weights. Required if the weight is placed on the left-side of the arrow and not placed first to be executed.
| |	Wraps a single weight for an edge. Required around arithmetic expressions (e.g. |5+1| or |x*2|). Optional otherwise.
->	Adds an edge between two nodes in a graph. Weight is defaulted to 1. Weight is always placed to the immediate right of the arrow.
<-	Adds an edge between two nodes in a graph. Weight is defaulted to 1. Weight can be placed to the immediate right or the immediate left of the arrow.
<->	Adds two edges between two nodes in a graph. Weights are defaulted to 1 on both sides. Can add weights like above.
,	Separates graph operations in recursion. The comma can be thought of as equivalent to “; `G: “`
`G: + A + B <-> A + C <- A, B ->5 C;`	Recursive example. What does this do? This adds Nodes A and B to the graph, adds a bidirectional edge between them with weights 1 each direction, then adds Node C to the graph and an edge from A to C with weight 1. Finally, an edge from B to C with weight 5 is added.

All operations can be cascaded together like in the last example, or they can be separated among multiple statements. Now for a deeper dive:

8.2.0.3 Add Node to Graph

```
Node A = Node(“test”);

Graph G;
```

```
G: + A;
```

8.2.0.4 Add Nodes to Graph

```
Node A = Node("5")

Node B = Node("6")

Node nodes[2];

nodes[0] = A;

nodes[1] = B;

Graph G;

G: + nodes[0] + node[1]; /* could wrap in a while loop as well */
```

8.2.0.5 Add Edge to Graph

```
Graph G;
G: h ->5 e;
```

Note: Graphs are mutable, so this modifies graph G in place!

8.2.0.6 Graph & Node Accessor Functions at a Glance

G.weight[A, B];	Returns the weight from node A to B in Graph G. returns -100000 if non existent.
A.name;	Returns the name of node A (as a string).
G.num_nodes;	Returns the number of nodes in Graph G (as an int).
G.num_neighbors[A];	Returns the number of neighbors to node A in Graph G (as an int).
G.node[n];	Returns the nth Node (index 0) in graph G. Segfaults if n > G.num_nodes.
G.neighbor[A, n];	Returns the nth neighbor to Node A in Graph G. Segfaults if n > G.num_neighbors[A].
A.curr_dist;	Returns the current distance stored in Node A. Used during graph algorithms.
A.visited;	Returns a bool whether Node A has been visited or not during a graph algorithm.

8.2.0.7 Get an Edge's weight from a Graph

Given a Graph G with an Edge that connects Node A and Node B:

```
int x = G.weight[A, B];
```

This returns the weight between node A and B in graph G.

8.2.0.8 Conditional Statements

The forms of conditional statements supported are:

```
if (expression) { statement }

if (expression) { statement } else { statement }
```

8.3 Return Statement

A function returns to its caller through the return statement which has the following forms:

```
return;

return (expression);
```

8.4 null Statement

The null statement is simply:

```
;
```

8.5 import Statement

The import statement adds modularity to YAGL and can be used to recycle code. The import statement(s) is typically used at the beginning of a YAGL file. It is used to import definitions and functionality from an existing YAGL file. Imports act similar to C's `#include` preprocessing directive in which the import statement is replaced with the source code of another .ygl file. To import the standard libraries of YAGL, simply place the following two lines of code at the top of your file:

```
import stdgraph.ygl

import stdalgo.ygl
```

And place stdgraph.ygl and stdalgo.ygl in the directory you run your YAGL program from. Note that imports are not terminated with a semicolon. This was simply a syntax and language design choice made to emulate C's preprocessing directives that do not require a semicolon at the end.

9 Control Flow and Scope

Programs are executed from top to bottom, but you are able to loop (while) or skip (if/else) blocks of code based on a condition. Also, programs include variables that are accessible depending on where they are declared.

9.1 if/else statement

An if/else statement takes a condition. If the condition is true, then the block of code inside the curly braces that follows the "if" is executed. If it is false, the block of code is skipped and the block of inside the curly braces following the "else" is executed. Users are given the option to omit the "else".

```
if { }

else { }
```

9.2 while loop

A while loop checks a condition and if it is true, then the block of code inside the curly braces is executed. The program goes back to the top of the loop and checks the condition again. If it is still true, the execution is repeated. This goes on until the condition is false and you break out of the loop.

```
while( boolean expression) { }
```

9.3 scope

Curly braces create a new scope. Variables that are declared outside any curly braces are available to be used anywhere in the program. Otherwise, variables are only accessible inside the curly braces. For example, if the declaration takes place inside a function, then the variable isn't available to other functions. This is true for if/else statements, while loops, and any other set of curly braces that create a new block and scope. Variables declared in outer blocks are in scope for inner, nested blocks, but variables declared in inner, nested blocks are not available to be accessed by outer scopes. These rules follow C's scoping rules.

10 Library Functions (Standard Library)

Below are some of the kinds of functions we plan on implementing using our Graph primitive operators to build the “Standard Graph Library” of our language. These functions are commonly used in many graph problems.

10.1 stdgraph.ygl

10.1.0.1 Copy Graph

`Graph copy_graph_lib(Graph A)`

Copies Graph A and returns the newly created graph.

```
Graph copy_graph_lib(Graph g) {

        Graph g2;

        int nodes = g.num_nodes;
        int curr = 0;
        while (curr < nodes) {
                Node current = g.node[curr];
                g2: + current;
                curr = curr + 1;
        }

        curr = 0;
        while (curr < nodes) {

                Node current = g.node[curr];
                int neighs = g.num_neighbors[current];
                int on = 0;
                while (on < neighs) {
                        Node to;
                        to  = g.neighbor[current, on];
                        int val = g.weight[current, to];
                        g2: current ->val to;
                        on = on + 1;
                }

                curr = curr + 1;
        }

        return g2;

}
```

*Please note that the above YAGL code block may include incomplete or inaccurate code highlighting.

10.1.0.2 Reverse Graph Edges

Copies Graph A and returns the newly created graph with all edges reversed.

```
Graph reverse_graph_lib(Graph g) {

        Graph g2;

        int nodes = g.num_nodes;
        int curr = 0;
        while (curr < nodes) {
                Node current = g.node[curr];
                g2: + current;
                curr = curr + 1;
        }

        curr = 0;
        while (curr < nodes) {

                Node current = g.node[curr];
                int neighs = g.num_neighbors[current];
                int on = 0;
                while (on < neighs) {
                        Node to;
                        to  = g.neighbor[current, on];
                        int val = g.weight[current, to];
                        g2: to ->val current; /* Reversed */
                        on = on + 1;
                }

                curr = curr + 1;
        }
        return g2;
}
```

*Please note that the above YAGL code block may include incomplete or inaccurate code highlighting.

10.1.0.3 Print Graph

Prints Graph A to stdout.

```
void print_graph_lib(Graph g) {
        int nodes = g.num_nodes;
```

```
        int curr = 0;
        String nodes_s = "Nodes: ";
        String edges = "Edges: ";
        while (curr < nodes) {
                Node current = g.node[curr];
                nodes_s = nodes_s + current.name + " ";
                curr = curr + 1;
                int neighs = g.num_neighbors[current];
                int on = 0;
                while (on < neighs) {
                        Node ne = g.neighbor[current, on];
                        String edge = current.name + " -> ";
                        edges = edges + edge + ne.name;
                        edges = edges + ", ";
                        on = on + 1;
                }
        }
        print(nodes_s);
        print(edges);
}
```

*Please note that the above YAGL code block may include incomplete or inaccurate code highlighting.

10.1.0.4 Graph's Linked List of Nodes (in C) to Array in YAGL

```
Node[1000] get_node_array(Graph G);
```

This will convert a Graph's internal linked list of nodes (in C), up to a size of 1000 nodes, to an Array type in YAGL.

```
Node[1000] get_node_array(Graph g) {
        Node[1000] nn;
        int size = g.num_nodes;
        int curr = 0;
        while (curr < size) {
                nn[curr] = g.node[curr];
                curr = curr + 1;
        }
        return nn;
}
```

*Please note that the above YAGL code block may include incomplete or inaccurate code highlighting.

10.2 stdalgo.ygl

10.2.0.1 Reset Graph (helper function)

`Void reset(Graph g)`

This function resets the attributes of g's nodes to an initialized state. Specifically, it resets every node's visited boolean to false and curr_dist variable to 0.

```
void reset(Graph g) {
    int size = g.num_nodes;
    int curr = 0;

    while (curr < size) {
      Node current;
      current = g.node[curr];
      current.visited = false;
      current.curr_dist = 0;
      curr = curr + 1;
    }
}
```

***Please note that the above YAGL code block may include incomplete or inaccurate code highlighting.**

10.2.0.2 Depth First Search

`Void dfs(Graph G, Node A, int depth)`

Prints all nodes within the given depth while performing a DFS traversal on Graph G starting at Node A being limited by the input depth.

```
void dfs_helper(Graph g, Node vertex, int depth){
   if (vertex.visited == true) {
       return;
   }
   if (vertex.curr_dist > depth) {
       return;
   }
   vertex.visited = true;

   /* Do whatever you want to the vertex */
   if (vertex.curr_dist == 0) {
   } else {
        print(vertex);
   }

   int size = g.num_neighbors[vertex];
```

```
   int curr = 0;

   while (curr < size) {
      Node current;
      current = g.neighbor[vertex, curr];
      if (current.curr_dist == 0) {
          current.curr_dist = vertex.curr_dist + 1;
      }
      curr = curr + 1;
   }
   curr = 0;
   while (curr < size) {
      Node current;
      current = g.neighbor[vertex, curr];
      /*print(g.weight[current, vertex]);*/
      dfs_helper(g, current, depth);
      curr = curr + 1;
   }

}

void dfs(Graph g, Node vertex, int depth) {

        reset(g);
        dfs_helper(g, vertex, depth);

}
```

*Please note that the above YAGL code block may include incomplete or inaccurate code highlighting.

10.2.0.3 Get First Encountered Node at Specified Depth

`Node dfs(Graph G, Node A, Node break, int depth)`

This algorithm performs a DFS traversal of the graph and returns the first Node it encounters at the specified *depth*. Starts the DFS traversal at the Node vertex. The following depicts the main meat of the function:

```
Node get_first_node_at_depth_helper(Graph g, Node vertex, Node break, int
depth){
   if (vertex.visited == true) {
       return break;
   }

   if (depth == vertex.curr_dist) {
```

```
        return vertex;
    }
    vertex.visited = true;

    int size = g.num_neighbors[vertex];
    int curr = 0;

    while (curr < size) {
       Node current;
       current = g.neighbor[vertex, curr];
       if (current.curr_dist == 0) {
           current.curr_dist = vertex.curr_dist + 1;
       }
       curr = curr + 1;
    }
    curr = 0;
    Node new; /* just a place holder */
    while (curr < size) {
       Node current;
       current = g.neighbor[vertex, curr];
       new = get_first_node_at_depth_helper(g, current, break, depth);
       if (new == break) {
       } else {
         return new;
       }
       curr = curr + 1;
    }
    return new;
}
```

*Please note that the above YAGL code block may include incomplete or inaccurate code highlighting.

11 References

[1] Dennis M. Ritchie. C Reference Manual. https://www.bell-labs.com/usr/dmr/www/cman.pdf.

[2] Edwards, Stephen. ”Programming Language and Translators.” MicroC.

[3] Tutorials Point. https://www.tutorialspoint.com/cprogramming/c_data_types.htm.

[4] LLVM. https://llvm.moe/ocaml/Llvm.html

[5] Gained inspiration for final report format from:
http://www.cs.columbia.edu/~sedwards/classes/2012/w4115-fall/reports/Funk.pdf

12 Examples that Best Represent the Power of YAGL

12.1 Get Node at Certain Depth in Graph (Modifying standard library)

Calling `dfs_helper_node` on graph g, starting Node vertex, breaking at the Node break, and searching for a node at depth will return either the break node (if there does not exist a node at the requested depth) or will return a node at the requested depth. Depth is defined by minimum number of steps from the starting Node vertex to another node in the given graph.

```
/* modified lib function */
Node dfs_helper_node(Graph g, Node vertex, Node break, int depth){
   if (vertex.visited == true) {
       return break;
   }

   if (depth == vertex.curr_dist) {
       return vertex;
   }
   vertex.visited = true;

   int size = g.num_neighbors[vertex];
   int curr = 0;

   while (curr < size) {
      Node current;
      current = g.neighbor[vertex, curr];
      if (current.curr_dist == 0) {
          current.curr_dist = vertex.curr_dist + 1;
      }
      curr = curr + 1;
   }
   curr = 0;
   Node new(""); /* just a place holder */
   while (curr < size) {
      Node current;
      current = g.neighbor[vertex, curr];
      /*print(g.weight[current, vertex]);*/
      new = dfs_helper_node(g, current, break, depth);
      if (new == break) {
      } else {
        return new;
      }
```

```
      curr = curr + 1;
   }
   return new;
}

void reset(Graph g) {
    int size = g.num_nodes;
    int curr = 0;

    while (curr < size) {
      Node current;
      current = g.node[curr];
      current.visited = false;
      current.curr_dist = 0;
      curr = curr + 1;
    }
}

Node dfs_node(Graph g, Node vertex, Node b, int depth) {

        reset(g);
        return dfs_helper_node(g, vertex, b, depth);

}
```

*Please note that the above YAGL code block may include incomplete or inaccurate code highlighting.

12.2 Hello World

The following program prints “hello world!”.

```
Graph hello; /* declare graph */

String hello_world = ""; /* declare string holder */

/* declare nodes */
Node break("");
Node h("h");
Node e("e");
Node l("l");
Node l2("l");
Node o("o");
Node space(" ");
```

```
Node w("w");
Node o2("o");
Node r("r");
Node l3("l");
Node d("d");
Node excl("!");

/* populate graph */
hello: + h + e + l + l2 + o + space + w + o2 + r + l3 + d + excl,
         h -> e -> l -> l2 -> o -> space -> w -> o2 -> r -> l3 -> d ->
excl;

/* Generate array of nodes based on depth */
int num_nodes = hello.num_nodes;
Node[11] nodes;
int depth = 0;
while (depth < num_nodes + 1) {
        nodes[depth] = get_first_node_at_depth(hello, h, break, depth);
        depth = depth + 1;
}

/* Populate hello_world string from nodes */
depth = 0;
while (depth < num_nodes) {
        Node curr = nodes[depth];
        hello_world = hello_world + curr.name;
        depth = depth + 1;
}
/* Done! Hello world! */
print(hello_world);
```

*Please note that the above YAGL code block may include incomplete or inaccurate code highlighting.

12.3 Social Media Representation

The program written in YAGL below demonstrates many features of our language. We use three separate social networking services as our models for the three graphs. The nodes of the graphs are the people that are part of the networks; notice how we can include the same node in different networks.

Performing various manipulations and augmentations viewable in the code comments, we ultimately want to find a list of friends of friends (relationships with a max depth of 2).

```
import stdalgo.ygl
```

```
import stdgraph.ygl

/* Social Media Application */

Graph fb;         /* Facebook  */
Graph tw;         /* Twitter   */
Graph ln;         /* LinkedIn  */

/* People */
Node adam("Adam");
Node james("Jamie");
/* Typo on purpose for demonstration later */
Node jack("Jack");
Node tank("Shvetank");
Node edwards("Edwards");
Node tiffany("Tiffany");
Node buehler("Buehler");
Node michael1("Michael1");
Node michael2("Michael2");


/* Add people to Facebook's network */
fb: + adam + james + jack + tank;

/* Add edges between people to make them friends */
fb: adam <-> james <-> jack, tank <-> jack, tank <-> adam;
/* Facebook friends is a bidirectional relationship */

printString("Facebook's Network");
printGraph(fb);


/* Add people to Twitter's network and edges between them to add followers
*/
tw: + edwards + michael1 -> edwards,     /* Twitter followers is a one
directional relationship (not required to follow back) */
    + michael2 + tank <- edwards;        /* Nodes can be in multiple graphs
*/

printString("Twitter's Network");
printGraph(tw);


/* Add people to LinkedIn's graphs and edges between people to make
```

```
connections, LinkedIn connections are bidirectional */
ln: + tiffany + buehler + james + edwards + michael1 + michael2 + adam +
tank <-> james <-> adam <-> edwards <-> michael1 <-> michael2 <-> tiffany
<-> edwards,
    + jack, buehler <-> james <-> jack;

printString("LinkedIn's Network");
printGraph(ln);

/* Delete the random people from LinkedIn since they chose to delete their
accounts */
ln: - tiffany - buehler - michael2 - michael1;

printString("LinkedIn's Network - with deletion");
printGraph(ln);
print_break();

/* Update typo! fixes in all graphs */
james = "James";
printString("LinkedIn's Network with James fixed");
printGraph(ln);

print("Friends and Friends of Friends of James");
/* Find friends of friends */
dfs(fb, james, 2);           /* Should find tank */
print("Facebook's Network, for reference:");
print(fb);
```

*Please note that the above YAGL code block may include incomplete or inaccurate code highlighting.

13 Project Plan

13.1 Planning Process

Our planning process was fluid. We spent our first couple weeks as a team planning what domain-specific language we would like to pursue and setting an overarching goal. Then as development began, we learned to also have short-term plans to help accomplish our overall goal. Each week we would set goals that we aimed to complete--allowing for room of deviation--and discuss by the end of each week. Goals changed based on practicality, feedback from team members, and feedback from the teaching staff.

13.2 Specification Process

In a meeting of the great minds, we planned key features that we wanted to include in YAGL. We had a lot of ideas that we trimmed down over time to have the essential building blocks of our language. These building blocks are super useful in creating graphing algorithms as you can explore in our standard library files (stdgraph.ygl and stdalgo.ygl).

13.3 Development Process

We developed independently using git branches and pull requests. Everyone would select features that were needed to be added and attempt to implement them. Each week we would meet to discuss each PR and merge the new feature's code into the main branch.

Adding a new feature required adding the required tokens to the scanner, parsing those tokens in the **yaglparser**, creating an abstract syntax tree which would be converted to a SAST that is semantically checked, and updating codegen to create the LLVM equivalent. As discussed later, each feature includes at least one happy path and one sad path test.

In short, our development process loosely followed principles of Agile.

13.4 Testing Process

YAGL was created with testing in mind. We attempted to adopt TDD (test-driven development) where we would create a test-file with a new feature. This new test file would undoubtedly fail at first. However, the test file was used as a check to see when the feature was successfully added. Once the test was passing, at least one additional sad-path test would be added, but in many cases another test was added. Tests are run automatically after the compiler is built with the **make test** or just the **make** command. See more in section 16.

13.5 Team Responsibilities

Our team used four roles: Team manager, Tester, Language guru, and System architect, to ensure development of YAGL. However, we did not attach these roles tightly and everyone played a role in every part of the compiler's development. We discussed almost every aspect of the compiler as a team and eventually came to an agreement on how to proceed. Each team member added full-blown features at each stage in development.

1. Adam Carpentieri - **Team Manager** - Emphasis on, but not limited to, ensuring progress was being made each week.
2. Jack Hurley - **Tester** - Emphasis on, but not limited to, ensuring code was sufficiently tested.
3. James Mastran - **Language Guru** - Emphasis on, but not limited to, ensuring scanner and parser was set up properly to match language specifications.

4. Shvetank Prakash - **System Architect** - Emphasis on, but not limited to, ensuring the coding environment was set up properly and the entire compilation flow worked.

13.6 Who Did What

As stated in the last section, we did not strictly adhere and limit ourselves to just the roles we were assigned at the beginning of the term. For the first half of the semester up and through the Hello World Milestone, we all worked together on lengthy Zoom calls to make sure we all understood how the pieces came together so that we were all on the same page.

After this point, we distributed the feature development in which every member of the team would take a feature off our TODO list and implement it from start to end (scanner → parser → ast → semantic checker → sast → codegen) and this included writing the tests as well for the feature. We would meet weekly then to merge our code & features together on a Zoom call.

This process worked well for us because we all wanted to learn about the *entire* compiler, and it helped us all contribute and succeed as a team in developing.

13.6.0.1 Adam's Features

Features Developed: Worked on float, string length, Node (including augments some c code), laid groundwork for attributes, Aided in adding Node data type.

13.6.0.2 Jack's Features

Features Developed: Worked on the edge operator and created the node attributes .visited and .curr_dist which we used to implement DFS in our own language.

13.6.0.3 James' Features

Features Developed: Declaration and assign on single line, String binops, Graph and Node data types (in our compiler as well as in C code), Added operations on Graphs, Added ability for chaining graph operations, Enhanced assessors (that Adam created) by providing the ability to add parameters to the accessor function.

13.6.0.4 Shvetank's Features

Features Developed: Arrays (completely native to YAGL, *no* C linking), Scoping via symbol table implementation for blocks, Preprocessor/Imports to provide modularity before invoking rest of compiler, Char data type, Bool data type, Assignments, No main function (all together), Strings (all together).

13.7 Project Timeline

This project encompassed the entire length of the Spring 2021 semester. Development was an ongoing process, but here are some key milestones:

- **Jan 20**
    - Team YAGL formed, graph language idea likely
- **Feb 3**
    - Proposal submitted
- **Feb 24**
    - LRM and working Parser submitted
- **Mar 24**
    - Hello World, showing a working String implementation, submitted
- **Apr 4**
    - Booleans
    - variable assignment
    - return statements
    - binop base included
    - Prevented user from defining a main function and calling main
- **Apr 7**
    - Graph and Node instantiation working
    - Added arrays
    - Added binops for Strings
    - Added declare and assign in a single line
    - Added if and while
- **Apr 9**
    - Added string length accessor
- **Apr 14**
    - One line edge instantiation working
    - Scoping
    - Added in-bounds checking for arrays
- **Apr 15**
    - Added remove node ability
- **Apr 18**
    - Added chaining of graph operations
- **Apr 21**
    - Library functions built
    - preprocessor/imports
- **Apr 24**
    - Added visited/curr_dist accessors to Nodes
    - Added advanced accessors to take arguments

    - generic printing
    - Added all other accessor functions
- **Apr 26** Final presentation made, project completion

13.8 Git log

The following is our git log. Note that some of the commits are lost since we used git branching. This gives you an idea of our progress. We had 31 branches we worked on in total.

Log:

```
c0c1f63  ShvetankPrakash  Mon Apr 26 23:20:44 2021 +0000    Final README updates and cleaned up Makefile.
97400a9             ShvetankPrakash  Mon Apr 26 22:15:33 2021 +0000    Removed old stray EDGE code from scanner.
7b3fa3c  James Mastran    Mon Apr 26 15:48:21 2021 -0400    Added 3 tests
f3a84a8  Jack Hurley      Mon Apr 26 11:31:39 2021 -0400    Merge branch 'main' of
https://github.com/mastranj/YAGL into main
04f5553 Adam7288         Mon Apr 26 14:11:22 2021 -0400    Add files via upload
dc65270Shvetank Prakash Mon Apr 26 10:54:39 2021 -0700    Rename yagl_one_line.ygl to yagl_in_one_slide.ygl
222381d ShvetankPrakash  Mon Apr 26 17:49:23 2021 +0000    Merge branch 'main' of
https://github.com/mastranj/YAGL into main
e7e2faf  ShvetankPrakash  Mon Apr 26 17:48:46 2021 +0000    Cleaned up unnecessary comments.
44d8a00             Shvetank Prakash Mon Apr 26 10:37:24 2021 -0700    Delete 4115_project_backup directory
4f004a1  Jack Hurley      Mon Apr 26 11:29:53 2021 -0400    updated README
4585a17 James Mastran    Mon Apr 26 09:26:31 2021 -0400    Removed commented out line
4ea5b59James Mastran     Mon Apr 26 09:23:05 2021 -0400    Added simplyfying code
6fc9e74  ShvetankPrakash  Mon Apr 26 07:36:38 2021 +0000    Added fail test case if import forgotten.
b2902b9             ShvetankPrakash  Mon Apr 26 07:32:04 2021 +0000    Added and fixed import test case.
d308161 ShvetankPrakash  Mon Apr 26 07:10:18 2021 +0000    Created separate file that compiles and runs exe.
ccd7016  Shvetank Prakash Mon Apr 26 00:02:15 2021 -0700    Delete fail-array4.err
9b51344 Shvetank Prakash Mon Apr 26 00:01:42 2021 -0700    Delete fail-array4.ygl
905f43b Shvetank Prakash Sun Apr 25 23:58:55 2021 -0700    Rename compile_and_run_single.sh to yagl_debugger
bef4ae4  James Mastran    Sun Apr 25 22:57:13 2021 -0400    Updated spacing and headers
8230663             James Mastran    Sun Apr 25 21:09:16 2021 -0400    Merge branch 'main' of
https://github.com/mastranj/YAGL into main
3056e65             James Mastran    Sun Apr 25 21:09:05 2021 -0400    rename name
7bd8bb8ShvetankPrakash  Mon Apr 26 00:53:08 2021 +0000    Removed old comment.
71ac2a6  James Mastran    Sun Apr 25 20:21:20 2021 -0400    added scoping
047252e             James Mastran    Sun Apr 25 20:18:54 2021 -0400    addedscoping
6bb457eJames Mastran     Sun Apr 25 20:14:43 2021 -0400    std lib
b2337cc James Mastran    Sun Apr 25 18:13:53 2021 -0400    std lib
233641d James Mastran    Sun Apr 25 17:44:11 2021 -0400    new test
7ce0efe  James Mastran    Sun Apr 25 15:04:34 2021 -0400    alpha renaming
83c74ae James Mastran    Sun Apr 25 15:03:32 2021 -0400    added demos in demos folder and made stdlibs
41a1160   mastranj Sun Apr 25 14:35:30 2021 -0400    Merge pull request #31 from mastranj/import
bcc8e51 mastranj Sun Apr 25 14:34:44 2021 -0400    Merge pull request #30 from mastranj/type_check_attribute
70ac433 mastranj Sun Apr 25 14:34:35 2021 -0400    Merge pull request #29 from mastranj/scope-fail-tests
a884235mastranj Sun Apr 25 14:33:23 2021 -0400    Merge pull request #28 from mastranj/struct_type
f7bc8d5  ShvetankPrakash  Sun Apr 25 18:29:47 2021 +0000    One more fix for deleting preprocessed files.
aa940e6ShvetankPrakash  Sun Apr 25 18:25:11 2021 +0000    Fixed import for fails.
efa0219  James Mastran    Sun Apr 25 13:21:46 2021 -0400    Added tests and unops
2c16299 James Mastran    Sun Apr 25 12:46:47 2021 -0400    Added tests
f10aff4   James Mastran    Sun Apr 25 11:43:10 2021 -0400    Changed name
4984f57 James Mastran    Sun Apr 25 11:18:22 2021 -0400    removed pointer
9edfd57 James Mastran    Sun Apr 25 11:15:53 2021 -0400    added new hello world
4618da2 James Mastran    Sun Apr 25 11:03:30 2021 -0400    decl and assign node in one go
580a621 James Mastran    Sun Apr 25 10:56:49 2021 -0400    Fixed return types and lib func
4a0e169 James Mastran    Sun Apr 25 10:35:20 2021 -0400    Fixed edge stuff
e43c77f  James Mastran    Sun Apr 25 09:48:27 2021 -0400    added reverse graph edges lib func
```

16040c8 James Mastran Sun Apr 25 09:43:37 2021 -0400 added better type check
d507b3b ShvetankPrakash Sun Apr 25 11:34:25 2021 +0000 Added preprocessing directive/import functionality to compiler.
84a7f02 James Mastran Sun Apr 25 01:18:42 2021 -0400 added get_node_array
ca1d558 James Mastran Sat Apr 24 22:31:07 2021 -0400 added tests
e2e799a James Mastran Sat Apr 24 21:40:38 2021 -0400 Removed not used
0c79112 James Mastran Sat Apr 24 21:34:42 2021 -0400 Changed prec
24286f1 James Mastran Sat Apr 24 21:31:00 2021 -0400 Added copy graph stdlib
e3935cc James Mastran Sat Apr 24 19:53:13 2021 -0400 changed get graph size to llvm
18953b5 ShvetankPrakash Sat Apr 24 23:33:19 2021 +0000 Added fail scope tests.
a9ea53d James Mastran Sat Apr 24 18:03:34 2021 -0400 Added types correctly
88fe63e James Mastran Sat Apr 24 17:49:20 2021 -0400 Upated types
1c6c5c8 James Mastran Sat Apr 24 16:30:39 2021 -0400 Removed bfs
f112b48 James Mastran Sat Apr 24 16:23:22 2021 -0400 Fixed final demo{
33594eb James Mastran Sat Apr 24 15:28:17 2021 -0400 Merged new changes
9d31285 James Mastran Sat Apr 24 15:27:42 2021 -0400 Added type check for curr_dist
49b45ac James Mastran Sat Apr 24 15:25:28 2021 -0400 junk
ac725f6 mastranj Sat Apr 24 15:24:04 2021 -0400 Merge pull request #27 from mastranj/curr_dist
9419167 mastranj Sat Apr 24 15:23:55 2021 -0400 Merge branch 'main' into curr_dist
7d170c5 mastranj Sat Apr 24 15:14:28 2021 -0400 Merge pull request #26 from mastranj/add_node_type
78f2863 James Mastran Sat Apr 24 14:27:38 2021 -0400 added get neighbors
582aa6d James Mastran Sat Apr 24 14:12:09 2021 -0400 Can get neighboring nodes
1378177 James Mastran Sat Apr 24 12:26:10 2021 -0400 Added get name attribute
82174ee James Mastran Sat Apr 24 12:17:20 2021 -0400 Added accessing nodes
10b7c3b Jack Hurley Sat Apr 24 11:33:22 2021 -0400 implemented curr_dist and fixed visited tests
11ff8c0 James Mastran Fri Apr 23 20:00:29 2021 -0400 Added other stuff
eca8030 Jack Hurley Fri Apr 23 17:35:42 2021 -0400 added fail tests
37a0341 James Mastran Fri Apr 23 16:32:31 2021 -0400 added get graph size
ac9601b James Mastran Fri Apr 23 16:15:59 2021 -0400 Added tests
38c06bf Jack Hurley Fri Apr 23 16:07:51 2021 -0400 added a new test
e76197c James Mastran Fri Apr 23 16:04:21 2021 -0400 Added update node name
2491b63 Jack Hurley Fri Apr 23 15:56:00 2021 -0400 .visited working
d38ff59 Jack Hurley Fri Apr 23 14:01:07 2021 -0400 returning bool works
8c0cb36 James Mastran Sun Apr 18 18:25:26 2021 -0400 Fixed
0ec4cd2 mastranj Sun Apr 18 18:16:22 2021 -0400 Merge pull request #24 from mastranj/chain_edge
d68dea3 Shvetank Prakash Sun Apr 18 15:15:31 2021 -0700 Added final demo project.
ac582ee James Mastran Sun Apr 18 17:09:08 2021 -0400 Fixed tests
7327148 James Mastran Sun Apr 18 17:04:32 2021 -0400 Added COMMA
8641451 James Mastran Sun Apr 18 16:23:46 2021 -0400 Fixed remove
311d8f4 James Mastran Sun Apr 18 01:18:18 2021 -0400 Added another test
68c038b James Mastran Sun Apr 18 01:15:33 2021 -0400 Added tests
55b0a0b James Mastran Sun Apr 18 01:07:16 2021 -0400 Cleaned up lang
3dd8c01 James Mastran Sun Apr 18 01:04:07 2021 -0400 Added bidirectional weights
f7922ff James Mastran Sat Apr 17 22:50:18 2021 -0400 Added & for edge recursion
7223e1a James Mastran Sat Apr 17 22:16:52 2021 -0400 cleaned up a little
2b544af James Mastran Sat Apr 17 19:50:05 2021 -0400 Added bidir and reverse dir edges
dc6186b James Mastran Sat Apr 17 15:20:01 2021 -0400 Added test
cfb2058 James Mastran Sat Apr 17 15:04:05 2021 -0400 Added chained ops such as remove node and add node with edges intermixed
1ecf686 James Mastran Sat Apr 17 14:23:30 2021 -0400 Added chained +
de1c310 James Mastran Fri Apr 16 13:13:31 2021 -0400 Added more tests and fixed some minor syntax issues
363613c James Mastran Fri Apr 16 00:40:25 2021 -0400 Added tests and changed pretty print
d68ef24 James Mastran Thu Apr 15 23:29:45 2021 -0400 chained edges
275d26a James Mastran Thu Apr 15 17:18:04 2021 -0400 Added expr to edges
23303c7 James Mastran Thu Apr 15 15:58:10 2021 -0400 fixed all warnings
a53c6c5 mastranj Thu Apr 15 15:33:44 2021 -0400 Merge pull request #23 from mastranj/remove_node
9d639f3 Jack Hurley Thu Apr 15 11:48:32 2021 -0400 added tests for edge
8bc365d Adam7288 Thu Apr 15 13:23:01 2021 -0400 added .out files for float, node, and string length (attr) tests
7353bad James Mastran Thu Apr 15 01:54:59 2021 -0400 Added remove node
c2d3ee3 James Mastran Wed Apr 14 23:49:19 2021 -0400 Added default to 1 for edge weight. Added demo3
6e766a7 mastranj Wed Apr 14 23:12:20 2021 -0400 Merge pull request #22 from mastranj/edge
6b5da63 mastranj Wed Apr 14 23:12:07 2021 -0400 Merge branch 'main' into edge
9742b96 James Mastran Wed Apr 14 22:58:44 2021 -0400 Fixed merge
c676144 mastranj Wed Apr 14 22:51:43 2021 -0400 Merge pull request #21 from mastranj/scope

f3d6a7b mastranj Wed Apr 14 22:51:37 2021 -0400 Merge branch 'main' into scope
e823edb mastranj Wed Apr 14 22:43:47 2021 -0400 Merge pull request #18 from mastranj/char
672c49e mastranj Wed Apr 14 22:43:36 2021 -0400 Merge branch 'main' into char
f2caa5b mastranj Wed Apr 14 22:39:50 2021 -0400 Merge pull request #17 from mastranj/array
6798bf9 mastranj Wed Apr 14 22:39:37 2021 -0400 Merge branch 'main' into array
8caf084 Shvetank Prakash Wed Apr 14 19:29:07 2021 -0700 Finished complete implementation of scope rules.
4ae648e James Mastran Wed Apr 14 22:25:42 2021 -0400 Fixed tests
138b87e James Mastran Wed Apr 14 22:19:40 2021 -0400 Merged nodes and graphs. Added insert node. Changed code style
2881a01 mastranj Wed Apr 14 21:37:18 2021 -0400 Merge pull request #20 from mastranj/Node
4939fba mastranj Wed Apr 14 21:37:09 2021 -0400 Merge branch 'main' into Node
a9560d7 mastranj Wed Apr 14 21:15:58 2021 -0400 Merge pull request #16 from mastranj/test_suite
47de862 mastranj Wed Apr 14 21:15:43 2021 -0400 Merge pull request #19 from mastranj/graph
97d2c20 Shvetank Prakash Wed Apr 14 18:01:26 2021 -0700 Got working implementation of scope.
55b5e3c Jack Hurley Wed Apr 14 19:32:18 2021 -0400 Fixed parsing error
4d64691 Adam7288 Wed Apr 14 17:21:35 2021 -0400 printnode function added
ad16cf8 Adam7288 Wed Apr 14 17:07:12 2021 -0400 instantiate node
90be07c Jack Hurley Tue Apr 13 16:03:58 2021 -0400 Fixed compilation errors
ac1cb5b Jack Hurley Tue Apr 13 14:15:39 2021 -0400 changed args for insert_edge
26abdc3 Jack Hurley Tue Apr 13 13:35:45 2021 -0400 edges untested
209c1f7 Shvetank Prakash Tue Apr 13 00:56:04 2021 -0700 Implemented char type with printing of chars as well.
f5eebfe Shvetank Prakash Mon Apr 12 23:53:15 2021 -0700 Added bounds checking of ints.
7c76b20 Adam7288 Tue Apr 13 00:23:09 2021 -0400 change node to a binding_assign (not compiling at this time)
196a3f9 James Mastran Mon Apr 12 23:42:21 2021 -0400 Added create edge function for jack
79170fa James Mastran Mon Apr 12 23:17:38 2021 -0400 Added some more functions
0d1f61c James Mastran Mon Apr 12 21:52:24 2021 -0400 removed ambiguity
a91beb6 James Mastran Mon Apr 12 20:39:36 2021 -0400 Checks in print graph if nodes/edges exist
a9d160c James Mastran Mon Apr 12 20:32:00 2021 -0400 Cleaned up code and defined other types. Added onto stdlib.c
7081e1b James Mastran Mon Apr 12 17:39:27 2021 -0400 working kinda
89ad6ed James Mastran Mon Apr 12 17:11:57 2021 -0400 got it to somewhat work
dff7ac1 James Mastran Mon Apr 12 10:49:09 2021 -0400 init commit
dd1c2c1 Adam7288 Sun Apr 11 23:21:38 2021 -0400 define node type, struct and pointer
e48d6b9 James Mastran Sun Apr 11 18:07:53 2021 -0400 Fixed tests
a2ea338 James Mastran Fri Apr 9 22:52:03 2021 -0400 Updated print graph function
a57ff85 James Mastran Fri Apr 9 21:36:07 2021 -0400 Fixed warning in graph.c
b003c7e mastranj Fri Apr 9 20:47:21 2021 -0400 Merge pull request #15 from mastranj/cleanup_warns
725ff62 mastranj Fri Apr 9 20:46:13 2021 -0400 Merge pull request #14 from mastranj/binop_string2
d592ac5 James Mastran Fri Apr 9 20:45:29 2021 -0400 merged main
897f7c0 James Mastran Fri Apr 9 20:35:21 2021 -0400 Fixed some merge conflict
731e39e mastranj Fri Apr 9 20:27:42 2021 -0400 Merge pull request #13 from mastranj/string_length
639f75b mastranj Fri Apr 9 20:25:52 2021 -0400 Merge branch 'main' into string_length
183bdb5 James Mastran Fri Apr 9 19:53:24 2021 -0400 Added graph mock
9580c2a James Mastran Fri Apr 9 14:37:42 2021 -0400 removed printbig
7552270 James Mastran Thu Apr 8 20:54:27 2021 -0400 Cleaned up warnings
107c9ed James Mastran Thu Apr 8 20:46:28 2021 -0400 Added checking operation to be +
9138dee James Mastran Thu Apr 8 20:37:36 2021 -0400 Cleaned up code
68b0a0b James Mastran Thu Apr 8 20:35:56 2021 -0400 Changed printbig file
781aeca James Mastran Thu Apr 8 20:30:25 2021 -0400 Added another test
75a0c3a James Mastran Thu Apr 8 20:28:23 2021 -0400 Fixed string concatenation
abcd4d2 James Mastran Thu Apr 8 15:34:48 2021 -0400 Added demo
440855f Adam7288 Thu Apr 8 00:01:01 2021 -0400 string length working
8df5bda James Mastran Wed Apr 7 23:09:06 2021 -0400 commented out code
686efeb James Mastran Wed Apr 7 21:30:24 2021 -0400 fixed merge problems
931876d mastranj Wed Apr 7 21:19:38 2021 -0400 Merge pull request #8 from mastranj/assign_and_decl
5a8ac9d mastranj Wed Apr 7 21:19:31 2021 -0400 Merge branch 'main' into assign_and_decl
197ea51 mastranj Wed Apr 7 21:16:30 2021 -0400 Merge pull request #7 from mastranj/binop_string
27f3133 mastranj Wed Apr 7 21:16:06 2021 -0400 Merge branch 'main' into binop_string
98075db mastranj Wed Apr 7 21:11:22 2021 -0400 Merge pull request #9 from mastranj/while/if
463e354 mastranj Wed Apr 7 21:08:08 2021 -0400 Merge pull request #11 from mastranj/float
5d1197b mastranj Wed Apr 7 21:07:58 2021 -0400 Merge branch 'main' into float
37eca36 mastranj Wed Apr 7 21:06:38 2021 -0400 Merge pull request #12 from mastranj/array
0260f6d Shvetank Prakash Wed Apr 7 18:05:27 2021 -0700 Added flexibility to arrays and tests.
0210cb6 Adam7288 Wed Apr 7 17:27:54 2021 -0400 add test file

cfc9039 Adam7288 Wed Apr 7 17:25:01 2021 -0400 initial work on striing.length
63dcb41 Adam7288 Mon Apr 5 17:47:14 2021 -0400 add back printf function
8ecc089 Adam7288 Mon Apr 5 16:09:22 2021 -0400 fixed string internal representation of float and changed llvm data type to double
7350a58 Shvetank Prakash Mon Apr 5 01:50:50 2021 -0700 Got basics of arrays working all the way.
12069ba Jack Hurley Sun Apr 4 21:01:11 2021 -0400 implemented if and while
16650d4 Shvetank Prakash Sun Apr 4 16:28:34 2021 -0700 Added dummy tests and got arrays through sast.
489b7f2 James Mastran Sun Apr 4 19:19:03 2021 -0400 Changed sad path
28228c7 James Mastran Sun Apr 4 19:15:25 2021 -0400 Added tests
959f0f6 James Mastran Sun Apr 4 19:09:10 2021 -0400 Added declare and assign in one line
4e8a830 James Mastran Sun Apr 4 17:47:23 2021 -0400 Cleaned up codegen
3077f58 James Mastran Sun Apr 4 17:46:04 2021 -0400 Added string concat
3e96180 James Mastran Sun Apr 4 17:09:38 2021 -0400 First init... string concat works
f4c7a61 mastranj Sun Apr 4 12:04:27 2021 -0400 Merge pull request #6 from mastranj/user_defined_function_vs_main
06a01f3 mastranj Sun Apr 4 12:02:42 2021 -0400 Merge pull request #5 from mastranj/binop_start
6133530 mastranj Sun Apr 4 12:02:18 2021 -0400 Merge branch 'main' into binop_start
450f319 mastranj Sun Apr 4 12:00:06 2021 -0400 Merge pull request #4 from mastranj/return
c044629 mastranj Sun Apr 4 11:59:53 2021 -0400 Merge branch 'main' into return
e0b3412 James Mastran Sun Apr 4 11:57:35 2021 -0400 Fixed missing semicolon
7f9cb70 mastranj Sun Apr 4 11:56:05 2021 -0400 Merge pull request #3 from mastranj/assign
37b7869 mastranj Sun Apr 4 11:55:56 2021 -0400 Merge branch 'main' into assign
13c7eb5 mastranj Sun Apr 4 11:49:35 2021 -0400 Merge pull request #2 from mastranj/bool
3c7be1f mastranj Sun Apr 4 11:44:59 2021 -0400 Merge branch 'main' into bool
5a83a31 mastranj Sun Apr 4 11:41:07 2021 -0400 Merge pull request #1 from mastranj/float
f8a00d3 James Mastran Sun Apr 4 01:05:23 2021 -0400 Added tests to ensure I did not break error message for duplicate function name defined
213720f James Mastran Sun Apr 4 01:01:53 2021 -0400 Removed old code not needed now. Just code from this branch
ab02e00 James Mastran Sun Apr 4 00:53:15 2021 -0400 prevent user defined main, calling main, and added corresponding tests
cd55595 James Mastran Sat Apr 3 23:56:07 2021 -0400 Added binop base
1598398 James Mastran Sat Apr 3 23:14:34 2021 -0400 Added tests for return
60bbcd7 James Mastran Sat Apr 3 23:01:45 2021 -0400 Can undo: removes test output on make clean
1338a03 James Mastran Sat Apr 3 22:59:24 2021 -0400 added returns
c9759c5 James Mastran Sat Apr 3 22:46:20 2021 -0400 more appropriate name
109dc55 Shvetank Prakash Sat Apr 3 16:02:43 2021 -0700 Completed variable assignment.
db87698 Shvetank Prakash Sat Apr 3 15:38:14 2021 -0700 Fixed error message of fail test case to make sure it matches.
7eb5e2f Shvetank Prakash Sat Apr 3 15:23:55 2021 -0700 Completed boolean expressions and printing of them.
d88730e Adam7288 Wed Mar 31 23:07:50 2021 -0400 implemented float type
385c3d1 Adam7288 Wed Mar 31 23:04:05 2021 -0400 implemented float type
ed64fa6 Adam7288 Sun Mar 28 18:30:58 2021 -0400 test errors
94d64cc Adam7288 Sun Mar 28 18:09:09 2021 -0400 codegen work
282a098 Adam7288 Sun Mar 28 17:12:31 2021 -0400 Fdecl progress
4c0584c Shvetank Prakash Sun Mar 28 10:54:16 2021 -0700 Adding vdecls and fdecls started.
63b94c2 Shvetank Prakash Thu Mar 25 18:15:06 2021 -0700 Delete compile_and_run.sh
5444405 Shvetank Prakash Thu Mar 25 18:12:27 2021 -0700 Last changes before turning in HelloWorld.
b1866b9 James Mastran Wed Mar 24 12:06:58 2021 -0400 Edit README and make file
61d6dd5 Shvetank Prakash Sat Mar 20 15:39:32 2021 -0700 Fixed file extension in testall.sh.
2d26c3d Shvetank Prakash Sat Mar 20 15:37:45 2021 -0700 Finished test suite for Hello World.
f7300c1 Shvetank Prakash Sat Mar 20 15:07:04 2021 -0700 Changed print to printInt.
dd12157 Shvetank Prakash Sat Mar 20 15:02:21 2021 -0700 Finished HelloWorld with Strings
1236a04 Shvetank Prakash Sat Mar 20 13:32:37 2021 -0700 Completed HelloWorld with ints.
ab0d86e Shvetank Prakash Thu Mar 18 19:24:56 2021 -0700 Went through flow from start to end.
2c7d74f Shvetank Prakash Thu Mar 18 17:56:01 2021 -0700 Backup directory.
db87bc7 Shvetank Prakash Thu Mar 18 17:53:35 2021 -0700 Completed AST for HelloWorld.
aa1729d Adam7288 Sun Mar 14 17:42:33 2021 -0400 3-14 group session changes
d77e7ec Adam7288 Wed Feb 24 17:22:37 2021 -0500 Add files via upload
23884ab Adam7288 Wed Feb 24 17:16:38 2021 -0500 rename file
7b80535 Adam7288 Wed Feb 24 16:40:05 2021 -0500 Update Makefile
7042ab0 Adam7288 Wed Feb 24 16:31:22 2021 -0500 Update Makefile with rebranding
068cdd3 Adam7288 Wed Feb 24 16:29:38 2021 -0500 Create yaglparse.mly
d60f403 Shvetank Prakash Tue Feb 23 18:38:38 2021 -0800 Added -> with no literal.
39b9d1e Shvetank Prakash Mon Feb 22 10:20:55 2021 -0800 Fixed QMARK expression.

e27a8f6 Shvetank Prakash Sun Feb 21 13:56:56 2021 -0800 Round 3 of work to complete parser and define unambiguous grammar.
0b53215 James Mastran Sun Feb 21 13:03:24 2021 -0500 Added todos. Note: we have 1 shift reduce problem.
8542327James Mastran Sun Feb 21 12:49:25 2021 -0500 Added action for specified regexpr for edge production. Added variables at top of file.
873acdd Shvetank Prakash Sat Feb 20 22:53:22 2021 -0800 Round 2 of work.
a1bc67d Shvetank Prakash Thu Feb 18 16:06:34 2021 -0800 First round of initial work.
1a2d5cb Shvetank Prakash Thu Feb 18 13:16:16 2021 -0800 Cleaned up repo and added MicroC example for starting point.
8297450 Adam7288 Sun Feb 14 14:30:25 2021 -0500 Add files via upload
cf269cb mastranj Sun Feb 14 13:23:47 2021 -0500 Initial commit

13.9 Development Environment

The development environment requires the use of Professor Edward's docker image. The environment settings provided through the docker image are enough to develop in YAGL. Notably, the llc and cc compilers are a must have.

We coded everything in Vim and SublimeText.

13.10 Programming Style Guide

13.10.0.1 YAGL Style

1. Most code style should follow that of ideal C programming style guide
2. Precede tricky code snippets with a multi-line comment
3. Graph operations can be done in a long chain, but the recommended use is to initially add all nodes in a single line followed by the "," comma operator with edge operations following. For example:

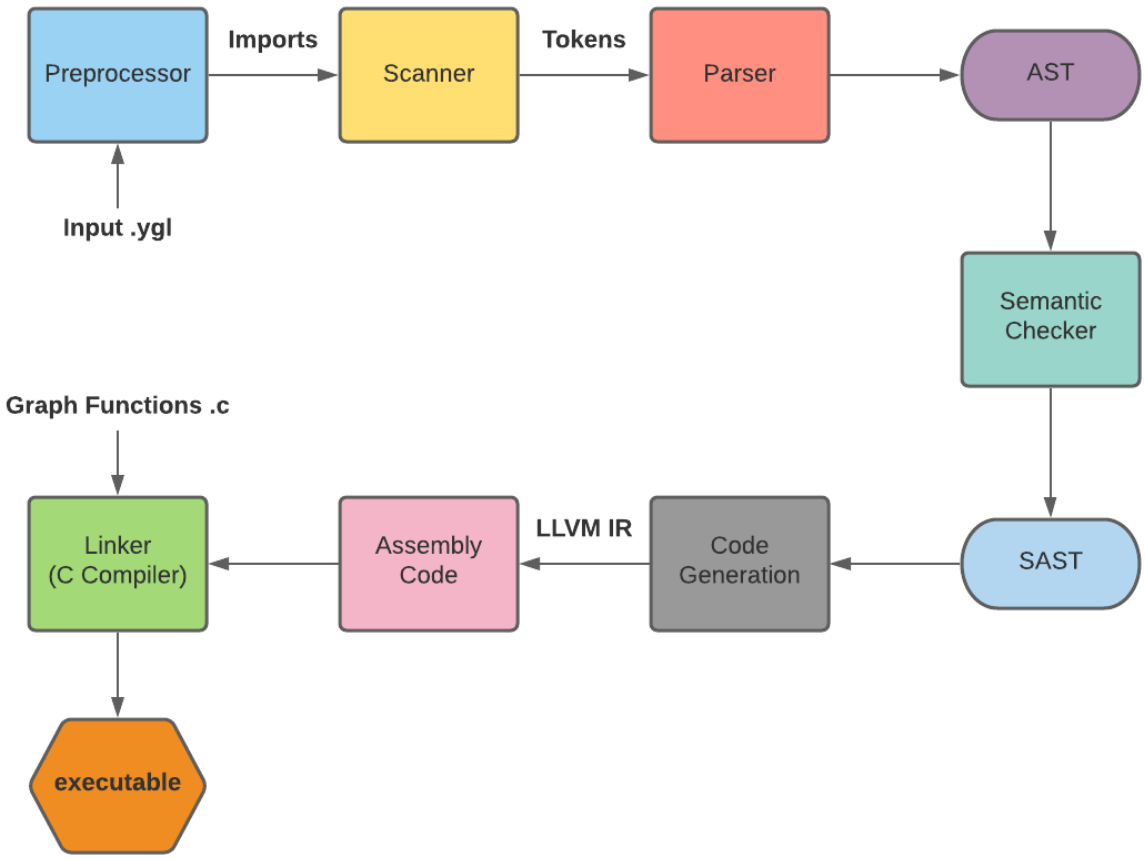
```
graph: node1 + node2 + node3 + … + nodeN,

       node1 -> node2 -> nodeN,

       node2 <-> node3,

       …
```

13.10.0.2 Group's Style in Ocaml

1. Try to keep lines small (< 80-100 characters)
2. Use consistent spacing with same matching block
3. Try to follow the style Professor Edward's showed us
4. Used proper indentation where appropriate
5. Comments above or to right of code referenced
6. Use self-documenting names for variables

7. Remove all warnings from compiler ASAP
8. Avoid using List.hd and List.tl
9. Use if-else only when applicable
    a. Prefer to use the match functionality if more than 2 branches

14 Compiler Architecture



Our compiler is broken down into a few stages. Our compiler uses the typical scanner to generate tokens that are parsed into an abstract syntax tree. The AST is converted to a semantically checked AST (SAST) and is converted into LLVM IR and Assembly code. The Assembly code is linked with our built-in functions to create an executable. What is unique to YAGL is the preprocessor (imports) and the linking of the .c file in the Linker (standard library). The preprocessor is described in the next section.

14.0.0.1 The Preprocessor (Shvetank)

YAGL's preprocessor searches the source code for import statements that include other .ygl files, and then it acts in a similar fashion to C's preprocessing directives. The preprocessor takes the code from the file to be imported and pastes it right where the import statement is made. This

modified file is then past to the scanner and rest of the compiler for analysis. This functionality allows for users to practice modularity which is a well established and encouraged practice when coding.

14.0.0.2 The Scanner (Adam, Jack, James, Shvetank)

After the preprocessor makes the correct .ygl source file, the scanner does lexical analysis on the input by splitting the input into a set of characters to generate tokens. This file is responsible for generating tokens such as, but not limited to, variable types (int, float, bool, Graph, ...), scope tokens, Strings, identifiers, and characters/escape characters.

14.0.0.3 The Parser (Adam, Jack, James, Shvetank)

The parser is fed this set of tokens to generate an abstract syntax tree (AST) which is defined by the rules of our grammar. The parser is given a set of rules that has a structure that reflects a context-free grammar.

14.0.0.4 The Semantics (Adam, Jack, James, Shvetank)

After an AST is created, the semantic checker ensures that the type of each node in the AST is an applicable type. Thus, after running the AST through the semantic checker, we are left with a Semantically-Checked Abstract Syntax Tree (SAST).

14.0.0.5 The Code Generator (Adam, Jack, James, Shvetank)

The code generator traverses through the SAST to create equivalent LLVM IR code and later Assembly code.

14.0.0.6 The Linking (Adam, Jack, James, Shvetank) & Built-ins (James)

Since graphs are complicated in nature, we used built-in functions. We declared these built-ins in codegen and then we link these C functions in the end with the assembly code generated throughout this process using the C compiler (cc).

15 Built-Ins (Graphs & Nodes Under-the-Hood)

Our built-in functions are written in C:

15.1 Graph *make_graph(int size)

This initializes a graph by malloc'ing enough room on the heap to hold a pointer to the linked-list of nodes, a pointer to the linked-list of edge lists, and other book-keeping variables.

15.2 Node *make_node(char *)

This initializes a node by malloc'ing enough room on the heap to hold a node. Nodes have 3 variables within them: an unique id (int), name (char *), curr_dist (int), and visited (bool). The name is set to the only argument provided.

15.3 void insert_node(struct Graph *, struct Node *)

This inserts a specified Node into a specified Graph. Graphs are dynamically sized so the user can insert as many Nodes without overfilling the Graph.

15.4 void remove_node(struct Graph *, struct Node *)

This removes a specified Node from the given Graph.

15.5 void insert_edge(struct Graph *g, struct Node *from, int weight, struct Node *to)

This inserts an edge that is from the from Node to the to Node with a specified weight inside a given Graph.

If the edge already exists, the weight is simply updated.

15.6 Node *get_neighbor(struct Graph *g, struct Node *n, int x)

This function will get the x-th node in Node n's edge list inside of Graph g. For example, if one calls get_neighbor(g, n, 5), this will get the 5th neighboring node (in the linked list of neighbors for Node n) of Node n in Graph g.

If a user tries to access a non-existent index (if x is greater than the number of nodes in a neighbor linked-list), a segfault will occur.

15.7 int get_num_neighbors(struct Graph *g, struct Node *n)

This function returns the number of neighbors a specified Node has in a given Graph.

15.8 Node *get_node(struct Graph *g, int n)

In Graph g, this will return the nth node in its linked-list for containing nodes. If a user tries to access a non-existent index (if n is greater than the number of nodes in a Graph), a segfault will occur.

15.9 int get_graph_size(struct Graph *g)

Returns how many nodes exist in a specified graph.

15.10 void print_graph(struct Graph *g)

Prints the given graph to standard output. We also have a standard library function (implemented in YAGL) that is a smaller version of this function.

15.11 char *sconcat(char *, char *)

Simply concatenates two strings and returns the concatenated string.

16 Testing Process

16.1 Automated Test Suite

Any file placed in the /tests directory that begins with “test-” or “fail-” will be executed as a happy-path or sad-path, respectively, test. Each test file has an accompanying “gold standard” output file (.out for happy-path tests and .err for sad-path tests) which is compared to the actual output of running the test source file.

Most of our tests are integration tests. However, some of the test files are small and test a single aspect of the compiler which could be deemed as unit tests.

Automated testing is run by typing **make test** in the root directory of our project where the Makefile is located.

16.2 Testing Overview

Every feature in YAGL was developed with Test-Driven-Development in mind. The feature-to-be-added would be given a new test that would originally fail. We would then develop the feature into YAGL until the test case passed. After that, a sad-path test would be added. Therefore, almost every feature in YAGL has at least one happy-path and one sad-path test. To give a comprehensive list:

- Arrays, chars, bools, etc.
- Assignment, declarations, and assignment-declaration in one line.
- Binops, Unops
- Main function testing (or lack-of a main function)
- Graph operations (declarations, adding nodes/edges, getting attributes)
- Import
- Scope, If/Else statements
- Return statements
- Built-in functions were tested

- The fibonacci recursive function was implemented as a test
- The gcd recursive function was implemented as a test
- Plus more!

16.2 Choosing Test Cases

Each individual member was responsible for writing the test cases for the feature they were developing. We tried to write numerous test cases for each feature, especially for the more complicated ones. Our mindset when trying to come up with test cases was first writing basic, normal cases, then edge cases, and then the most weird, “contrived” examples that a user may write to try and screw things up. We believe using this approach allowed us to write a very robust language. Check out our test cases in the appendix to see sample cases and how many we added!

17 Lessons Learned

17.1 Adam Carpentieri

As someone who has used and thought about programming languages for the better part of my life, it is embarrassing to admit that I never bothered to ask how the mechanics actually worked. Starting with the theory, in terms of NFA’s, to DFA’s, all the way through representations of basic blocks, one gains an appreciation for the solid theoretical foundations this topic is built on.

The project allowed us to see the connections between the practical and theoretical. It is fascinating how so many of the ideas used to generate machine code from client code use the exact same conventions as have been established since at least the 1970’s (Aho, et all).

The project crystallized our understanding of the subject matter, albeit in a disjointed chaotic way. This was advertised from the beginning of the class, so the instructor gets a pass.

I can honestly say that in my career with Columbia and its many group projects, this has been the best experience I’ve had so far. Chalk it up to luck, but I’ve never had the pleasure of working with more motivated, diligent, creative people thus far. No one needed to be managed, we all did a good amount, and at various times, we all carried the extra weight if one of us had something else going on from other classes. When successful people give the advice, “surround yourself with great people”, I now understand how important that can be.

17.2 Jack Hurley

The project was a great learning experience as I was able to see how all the different components of a compiler work together. I found that the best way to work on the compiler was by focusing on one feature at a time, starting at the scanner and working all the way through to

the codegen. When we first started, it was hard to see how the stages of the compiler fit together but after implementing our first feature, the other features were relatively easy to implement.

This was my first class where I had basically a semester-long project. I found that gauging how much progress we needed to be making throughout the semester the most difficult part. Thankfully we had a good manager and having weekly meetings was very important for us to stay on track. I thought our project came out great and had fun doing it.

17.3 James Mastran

For the last couple years of my undergraduate life, I had a vague understanding of how compilers worked (such as creating a bunch of tokens and trying to parse them into meaning); however, this class really shined the light on how much I did not know. I previously had an understanding of CS Theory (Regular Expressions, NFAs, etc.), but this course brought all these topics together. I now understand a full life cycle of a compiler!

The biggest overarching lessons I learned from the course include: coding in a functional language (OCaml), learning the parts of a compiler, reading assembly, and computing lambda functions.

The project was challenging. Not one member of our group had any prior knowledge on OCaml let alone any functional language. I can confidently say, we all learned OCaml pretty well. In addition, I have a better understanding of how the scanner, parser, AST, SAST (and semantic checker), and codegen work together. This understanding was cemented by the theoretical teachings during lecture and the practical aspects of the project. What really oppguned our group's knowledge was creating the LLVM assembly code via the codegen. While difficult, our group explored how to use build_store, build_load, and the likes to improve our project.

The members of our group were all open-minded, results-driven, and passionate. Any one of these qualities is rare to find in all members of a group. I feel very fortunate to have been on this team and to build such a cool language.

17.4 Shvetank Prakash

At a high level, what I gained most out of this class was a remarkable appreciation for compilers that I did not have before. I realize now how much I and many others probably take being able to run a “simple” command like `gcc foo.c` or clicking a green “play button” in an IDE for granted. After learning in this class just how complex a piece of software a compiler is from both lecture and first hand experience through the project, I now will never take compiling code for granted as a “trivial” task because I know how much work developing such a tool is. Moreover, I now feel that building these translators is such a significant and key accomplishment of our field that we under appreciate so much because they have provided a level of abstraction that has been

invaluable for the growth of our field. No wonder Professor Aho won the Turing Award for his tremendous work!

More specifically, I learned in this class in a very tangible way to see how the theory of computer science is grounded in real useful applications through the use of regular expressions for scanning tokens, context free grammars for syntax and parsing, and the concept of Turing machines when discussing a language's computability (i.e. if it is Turing Complete). I was taking CS Theory simultaneously with this class, so it was very nice to see the theoretical lectures in my CS Theory class being used and put into practice to build things just a couple weeks later each time.

Additionally, I learned really well how all the pieces of a compiler work together and the role of each part. It was interesting to see how each pass takes the previous representation and slightly transforms the code more and more until we finally get down to the machine code. I know we primarily talked more about the "frontend" of a typical compiler, in the sense that we did not get to talk in detail about how all the code optimizations and what not are done which I believe is considered the "backend", but I am now curious to learn more about this part as well after this class. I also really enjoyed learning LLVM which I got to intimately through the project as I know LLVM is an industry strength project. Some other topics I appreciated learning about were runtime environments since I never really understood what these were until this class and the beauty of functional programming, which I and my team had no prior experience with.

Lastly, I would just like to say as I reflect on this class that the project was truly enjoyable, and the team we had made such a challenging task much more pleasant. Prof. Edwards spent a large part of the very first lecture stressing how important selecting a team was, and I am grateful to have gotten so lucky as we did not even all know each other beforehand. It was a pleasure to work with such a great group of people.

18 Advice to Other Teams

- Dive into OCaml ASAP, learn it sooner than later. Once our group understood OCaml, our lives were much easier.
- Dive into adding features. You'll learn along the way with every painstaking mistake.
- Use your language to implement tough code that can be used over and over rather than only adding onto your language by adding features to the compiler.
- As all previous 4115 PLT Teams have said, start early: it will reflect in your final project's success!

19 Appendix (Our code)

19.1 scanner.mll (Adam, Jack, James, Shvetank)

```
(* Ocamllex scanner for YAGL *)

{ open Yaglparse }

let digit = ['0' - '9']
let digits = digit+
let ascii = [' ' - '~']
let escapeChars = ('\\' ['b' 't' 'r' 'n'])

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"     { comment lexbuf }           (* Comments *)
| '('      { LPAREN }
| ')'      { RPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
| "["      { LBRAC }
| "]"      { RBRAC }
| ';'      { SEMI }
| ','      { COMMA }
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '='      { ASSIGN }
| "=="     { EQ }
| '<'      { LT }
| ">"      { GT }
| "&&"     { AND }
| "||"     { OR }
| "|"      { BAR }
| "!"      { NOT }
| ":"      { COLON }
| "."      { DOT }
| "->"     { ARROW }
| "<-"     { REVARROW }
| "<->"    { BIARROW }
| "if"     { IF }
| "else"   { ELSE }
| "while"  { WHILE }
| "return" { RETURN }
```

```
| "int"    { INT }
| "bool"   { BOOL }
| "char"   { CHAR }
| "float"  { FLOAT }
| "Graph"  { GRAPH }
| "String" { STRING }
| "Node"   { NODE }
| "void"   { VOID }
| "true"   { BLIT(true)  }
| "false"  { BLIT(false) }
| digits as lxm { LITERAL(int_of_string lxm) }
| digits '.'  digit* ( ['e' 'E'] ['+' '-']? digits )? as lxm { FLIT(lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*     as lxm { ID(lxm) }
| "'" escapeChars "'"                                 as lxm { match
lxm.[2] with
                                                                 'b' ->
CHRLIT('\b')
                                                                |'t' ->
CHRLIT('\t')
                                                                |'r' ->
CHRLIT('\r')
                                                                |'n' ->
CHRLIT('\n')
                                                                | _ ->
raise (Failure "Internal error. Case should never be reached.")
                                                              }
| "'" ascii "'"                                       as lxm {
CHRLIT(String.get lxm 1) }
| '"' (( ascii # '"' )* escapeChars*)+ '"'            as lxm {
STRLIT(String.sub lxm 1 ((String.length lxm )-2) ) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

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

19.2 yaglparse.mly (Adam, Jack, James, Shvetank)

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

type op = Add | Sub | Mult | Div | Equal | Less | Greater |
          And | Or  | Link | RevLink | BiLink(* | Arrow | Colon *)
```

```
type uop = Neg | Not

type expr =
    Literal of int
  | FLit of string
  | BoolLit of bool
  | ChrLit of char
  | StrLit of string
  | Id of string
  | NodeLit of string * expr
  | GraphLit of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Assign of string * expr * expr
  | Call of string * expr list
  | Attr of string * string * expr * expr
  | Access of string * expr
  | EdgeList of expr * expr list
  | EdgeOp of expr * expr * op * expr * expr
  | EdgeOpBi of expr * expr * op * expr * expr * expr
  | NodeAttr of expr * string * expr
  | Noexpr

type typ = Void | Int | String | Float | Bool | Char | Array of typ * expr
         | Node | Edge| Graph

type bind = typ * string

type stmt =
    Block of stmt list
  | Expr of expr
  | If of expr * stmt * stmt
  | Bfs of expr * expr * expr * stmt
  | While of expr * stmt
  | Binding of bind      (* Only for vdecls *)
  | Binding_Assign of bind * expr
  | Return of expr

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

```
type program = stmt list * func_decl list

(* Pretty-printing functions *)

let string_of_op = function
    Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
  | Equal -> "=="
  | Less -> "<"
  | Greater -> ">"
  | And -> "&&"
  | Or -> "||"
  | Link -> "->"
  | RevLink -> "<-"
  | BiLink -> "<->"
  (*
  | Arrow -> "->"
  | Colon -> ":"
  *)

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

let rec string_of_expr = function
    Literal(l) -> string_of_int l
  | FLit(l) -> l
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | NodeLit(_, name) -> string_of_expr name
  | GraphLit(name) -> name
  | StrLit(str) -> str
  | ChrLit(c) -> Char.escaped c
  | Id(s) -> s
  | NodeAttr(e1, e2, e3) ->
      string_of_expr e1 ^ " " ^ e2 ^ " " ^ string_of_expr e3
  | Attr(s, a, e, e2) -> if e = Noexpr then
                                 s ^ "." ^ a
                         else ( if e2 = Noexpr then
                                  s ^ "." ^ a ^ "[" ^ string_of_expr e ^ "]"
                                else
```

```
                                s ^ "." ^ a ^ "[" ^ string_of_expr e ^ ",
" ^ string_of_expr e2 ^ "]"
                            )
  | 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, e1, e2) ->
      (match e2 with
         Noexpr -> v ^ " = " ^ string_of_expr e1
       | _ -> v ^ "[" ^ string_of_expr e1 ^ "]" ^ " = " ^ string_of_expr e2
      )
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | Access(id, e) -> id ^ "[" ^ string_of_expr e ^ "]"
  | Noexpr -> ""
  | EdgeOpBi(_, e1, o, e3, e4, e5) -> string_of_expr e1 ^ " "^
string_of_expr e5 ^ string_of_op o ^ "|"
        ^ string_of_expr e3 ^ "| " ^ string_of_expr e4
  | EdgeOp(_, e1, o, e3, e4) -> string_of_expr e1 ^ " " ^ string_of_op o ^
"|"
        ^ string_of_expr e3 ^ "| " ^ string_of_expr e4
  | EdgeList(e1, e2) -> string_of_expr e1 ^ ": "
        ^ (List.fold_left (fun s e -> s ^ match e with
               EdgeOp(_, e1, o, e3, e4) -> (match o with
                       Link -> (string_of_expr e1 ^ " " ^ string_of_op o ^
"|"
                        ^ string_of_expr e3 ^ "| ")
                     | _ -> "[" ^ string_of_op o ^ " " ^ string_of_expr e4
^ "],  ")
             | EdgeOpBi(_, e1, o, e3, _, e5) ->
                       (string_of_expr e1 ^ " |" ^ string_of_expr e5
                        ^ "|" ^ string_of_op o ^ "|"
                        ^ string_of_expr e3 ^ "| ")
             | _ -> ""
        ) "" (List.rev e2))
        ^ match (List.hd e2) with
               EdgeOp(_, _, _, _, e4) -> string_of_expr e4
             | EdgeOpBi(_, _, _, _, e4, _) -> string_of_expr e4
             | _ -> ""
  (*| NodeOfGraph(e1, e2) -> "(" ^  e1 ^ ", "
    ^ e2 ^ ")"
  *)

let rec string_of_typ = function
```

```
    Void        -> "void"
  | Int         -> "int"
  | Float       -> "float"
  | String      -> "String"
  | Bool        -> "bool"
  | Char        -> "char"
  | Node        -> "Node"
  | Graph       -> "Graph"
  | Edge        -> "Edge"
  | Array(t, e) -> string_of_typ t ^ "[" ^ string_of_expr e ^ "]"

let rec string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^
string_of_stmt s
  | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
      string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
  | Bfs(e1, e2, e3, s) ->
      "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ; " ^
      string_of_expr e3  ^ ") " ^ string_of_stmt s
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s
  | Binding(t, id) -> string_of_typ t ^ " " ^ id ^ ";\n"
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n"
  | Binding_Assign((t, id), e) ->
        match e with
                Assign(_, e',_) -> string_of_typ t ^ " " ^ id ^ " = "
                                   ^ string_of_expr e' ^ ";\n"
                | _           -> string_of_typ t ^ " " ^ id ^ " = "
                                   ^ string_of_expr e  ^ ";\n"

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

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

19.3 ast.ml (Adam, Jack, James, Shvetank)

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

type op = Add | Sub | Mult | Div | Equal | Less | Greater |
          And | Or  | Link | RevLink | BiLink(* | Arrow | Colon *)

type uop = Neg | Not

type expr =
    Literal of int
  | FLit of string
  | BoolLit of bool
  | ChrLit of char
  | StrLit of string
  | Id of string
  | NodeLit of string * expr
  | GraphLit of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Assign of string * expr * expr
  | Call of string * expr list
  | Attr of string * string * expr * expr
  | Access of string * expr
  | EdgeList of expr * expr list
  | EdgeOp of expr * expr * op * expr * expr
  | EdgeOpBi of expr * expr * op * expr * expr * expr
  | NodeAttr of expr * string * expr
  | Noexpr

type typ = Void | Int | String | Float | Bool | Char | Array of typ * expr
         | Node | Edge| Graph

type bind = typ * string

type stmt =
    Block of stmt list
  | Expr of expr
  | If of expr * stmt * stmt
  | Bfs of expr * expr * expr * stmt
  | While of expr * stmt
  | Binding of bind      (* Only for vdecls *)
  | Binding_Assign of bind * expr
  | Return of expr
```

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

type program = stmt list * func_decl list

(* Pretty-printing functions *)

let string_of_op = function
    Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
  | Equal -> "=="
  | Less -> "<"
  | Greater -> ">"
  | And -> "&&"
  | Or -> "||"
  | Link -> "->"
  | RevLink -> "<-"
  | BiLink -> "<->"
  (*
  | Arrow -> "->"
  | Colon -> ":"
  *)

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

let rec string_of_expr = function
    Literal(l) -> string_of_int l
  | FLit(l) -> l
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | NodeLit(_, name) -> string_of_expr name
  | GraphLit(name) -> name
  | StrLit(str) -> str
  | ChrLit(c) -> Char.escaped c
  | Id(s) -> s
  | NodeAttr(e1, e2, e3) ->
```

```
      string_of_expr e1 ^ " " ^ e2 ^ " " ^ string_of_expr e3
  | Attr(s, a, e, e2) -> if e = Noexpr then
                            s ^ "." ^ a
                        else ( if e2 = Noexpr then
                               s ^ "." ^ a ^ "[" ^ string_of_expr e ^ "]"
                               else
                               s ^ "." ^ a ^ "[" ^ string_of_expr e ^ ", " ^
string_of_expr e2 ^ "]"
                        )
  | 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, e1, e2) ->
      (match e2 with
         Noexpr -> v ^ " = " ^ string_of_expr e1
       | _ -> v ^ "[" ^ string_of_expr e1 ^ "]" ^ " = " ^ string_of_expr e2
      )
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | Access(id, e) -> id ^ "[" ^ string_of_expr e ^ "]"
  | Noexpr -> ""
  | EdgeOpBi(_, e1, o, e3, e4, e5) -> string_of_expr e1 ^ " "^ string_of_expr e5 ^
string_of_op o ^ "|"
        ^ string_of_expr e3 ^ "| " ^ string_of_expr e4
  | EdgeOp(_, e1, o, e3, e4) -> string_of_expr e1 ^ " " ^ string_of_op o ^ "|"
        ^ string_of_expr e3 ^ "| " ^ string_of_expr e4
  | EdgeList(e1, e2) -> string_of_expr e1 ^ ": "
        ^ (List.fold_left (fun s e -> s ^ match e with
                EdgeOp(_, e1, o, e3, e4) -> (match o with
                        Link -> (string_of_expr e1 ^ " " ^ string_of_op o ^ "|"
                         ^ string_of_expr e3 ^ "| ")
                      | _ -> "[" ^ string_of_op o ^ " " ^ string_of_expr e4 ^ "],  ")
              | EdgeOpBi(_, e1, o, e3, _, e5) ->
                        (string_of_expr e1 ^ " |" ^ string_of_expr e5
                         ^ "|" ^ string_of_op o ^ "|"
                         ^ string_of_expr e3 ^ "| ")
              | _ -> ""
        ) "" (List.rev e2))
        ^ match (List.hd e2) with
                EdgeOp(_, _, _, _, e4) -> string_of_expr e4
              | EdgeOpBi(_, _, _, _, e4, _) -> string_of_expr e4
              | _ -> ""
  (*| NodeOfGraph(e1, e2) -> "(" ^  e1 ^ ", "
    ^ e2 ^ ")"
```

```
  *)

let rec string_of_typ = function
    Void        -> "void"
  | Int         -> "int"
  | Float       -> "float"
  | String      -> "String"
  | Bool        -> "bool"
  | Char        -> "char"
  | Node        -> "Node"
  | Graph       -> "Graph"
  | Edge        -> "Edge"
  | Array(t, e) -> string_of_typ t ^ "[" ^ string_of_expr e ^ "]"

let rec string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt s
  | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
      string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
  | Bfs(e1, e2, e3, s) ->
      "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ; " ^
      string_of_expr e3  ^ ") " ^ string_of_stmt s
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s
  | Binding(t, id) -> string_of_typ t ^ " " ^ id ^ ";\n"
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n"
  | Binding_Assign((t, id), e) ->
        match e with
                Assign(_, e',_) -> string_of_typ t ^ " " ^ id ^ " = "
                                   ^ string_of_expr e' ^ ";\n"
                | _           -> string_of_typ t ^ " " ^ id ^ " = "
                                   ^ string_of_expr e  ^ ";\n"

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

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

19.4 sast.ml (Adam, Jack, James, Shvetank)

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

open Ast

type sexpr = typ * sx
and sx =
    SLiteral of int
  | SFLit of string
  | SBoolLit of bool
  | SChrLit of char
  | SStrLit of string
  | SId of string
  | SNodeLit of string * sexpr
  | SGraphLit of string
  | SBinop of sexpr * op * sexpr
  | SNodeAttr of sexpr * typ * sexpr
  | SUnop of uop * sexpr
  | SAssign of string * sexpr * sexpr
  | SAssignNode of string * sexpr * sexpr
  | SCall of string * sexpr list
  | SAttr of sexpr * string * sexpr * sexpr
  | SAccess of string * sexpr
  | SEdgeList of sexpr * sexpr list
  | SEdgeOp of sexpr * sexpr * op * sexpr * sexpr
  | SEdgeOpBi of sexpr * sexpr * op * sexpr * sexpr * sexpr
  | SNoexpr

type sstmt =
    SBlock of sstmt list
  | SExpr of sexpr
  | SIf of sexpr * sstmt * sstmt
  | SBfs of sexpr * sexpr * sexpr * sstmt
  | SWhile of sexpr * sstmt
  | SBinding of bind
  | SBinding_Assign of bind * sexpr
  | SReturn of sexpr

type sfunc_decl = {
    styp : typ;
    sfname : string;
```

```
    sformals : bind list;
    sbody : sstmt list;
  }

type sprogram = sfunc_decl list

(* Pretty-printing functions *)

let rec string_of_sexpr (t, e) =
  "(" ^ string_of_typ t ^ " : " ^ (match e with
    SLiteral(l) -> string_of_int l
  | SFLit(l) -> l
  | SBoolLit(true) -> "true"
  | SBoolLit(false) -> "false"
  | SChrLit(c) -> Char.escaped c
  | SNodeLit(_, name)  -> string_of_sexpr name
  | SGraphLit(name) -> name
  | SStrLit(str) -> str
  | SId(s) -> s
  | SNodeAttr(e1, t, e2) ->
      string_of_sexpr e1 ^ " " ^ string_of_typ t ^ " " ^  string_of_sexpr e2
  | SAttr(sx, a, e, e2) -> (match e with
                        (_, SNoexpr) | (Void, _) -> string_of_sexpr sx ^ "." ^ a
                      | _ -> string_of_sexpr sx ^ "." ^ a ^ "[" ^ string_of_sexpr e
                           ^ ", " ^ string_of_sexpr e2 ^ "]"  )
  | SBinop(e1, o, e2) ->
      string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_sexpr e2
  | SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
  | SAssign(v, e1, e2) | SAssignNode(v, e1, e2) ->
      (match e2 with
          (Void, SNoexpr) -> v ^ " = " ^ string_of_sexpr e1
        | _ -> v ^ "[" ^ string_of_sexpr e1 ^ "]" ^ " = " ^ string_of_sexpr e2
      )
  | SCall(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^ ")"
  | SAccess(id, e) -> id ^ "[" ^ string_of_sexpr e ^ "]"
  | SNoexpr -> ""
  | SEdgeOpBi(_, e1, o, e3, e4, e5) -> string_of_sexpr e1 ^ " " ^ string_of_sexpr e5 ^
string_of_op o ^ "|"
        ^ string_of_sexpr e3 ^ "| " ^ string_of_sexpr e4
  | SEdgeOp(_, e1, o, e3, e4) -> string_of_sexpr e1 ^ " " ^ string_of_op o ^ "|"
        ^ string_of_sexpr e3 ^ "| " ^ string_of_sexpr e4
  | SEdgeList(e1, e2) -> string_of_sexpr e1 ^ ": "
        ^ (List.fold_left (fun s e -> s ^ match e with
```

```
                (_, SEdgeOp(_, e2, o, e3, e4)) -> (match o with
                        Link -> (string_of_sexpr e2 ^ " " ^ string_of_op o
                         ^ string_of_sexpr e3 ^ " ")
                      | _ -> "[" ^ string_of_op o ^ " " ^ string_of_sexpr e4 ^ "] ")
              |  (_, SEdgeOpBi(_, e2, o, e3, _, e5)) ->
                        (string_of_sexpr e2 ^ " " ^ string_of_sexpr e5 ^ string_of_op o
                         ^ string_of_sexpr e3 ^ " ")
              | _ -> ""
        ) "" (List.rev e2))
        ^ match (List.hd e2) with
                (_, SEdgeOp(_, _, _, _, e4)) -> string_of_sexpr e4
              | (_, SEdgeOpBi(_, _, _, _, e4, _)) -> string_of_sexpr e4
              | _ -> ""
                          ) ^ ")"

let rec string_of_sstmt = function
    SBlock(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_sstmt stmts) ^ "}\n"
  | SExpr(expr) -> string_of_sexpr expr ^ ";\n";
  | SIf(e, s, SBlock([])) ->
      "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s
  | SIf(e, s1, s2) ->  "if (" ^ string_of_sexpr e ^ ")\n" ^
      string_of_sstmt s1 ^ "else\n" ^ string_of_sstmt s2
  | SBfs(e1, e2, e3, s) ->
      "for (" ^ string_of_sexpr e1  ^ " ; " ^ string_of_sexpr e2 ^ " ; " ^
      string_of_sexpr e3  ^ ") " ^ string_of_sstmt s
  | SWhile(e, s) -> "while (" ^ string_of_sexpr e ^ ") " ^ string_of_sstmt s
  | SBinding(t, id) -> "(" ^ string_of_typ t ^ " : " ^ id ^ ");\n"
  | SBinding_Assign((t, id), e) ->
        string_of_typ t ^ " " ^ id ^ " = " ^ string_of_sexpr e ^ ";\n"
  | SReturn(e) -> "return " ^ string_of_sexpr e ^ ";\n"

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

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

19.5 semant.ml (Adam, Jack, James, Shvetank)

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

```
open Ast

type sexpr = typ * sx
and sx =
    SLiteral of int
  | SFLit of string
  | SBoolLit of bool
  | SChrLit of char
  | SStrLit of string
  | SId of string
  | SNodeLit of string * sexpr
  | SGraphLit of string
  | SBinop of sexpr * op * sexpr
  | SNodeAttr of sexpr * typ * sexpr
  | SUnop of uop * sexpr
  | SAssign of string * sexpr * sexpr
  | SAssignNode of string * sexpr * sexpr
  | SCall of string * sexpr list
  | SAttr of sexpr * string * sexpr * sexpr
  | SAccess of string * sexpr
  | SEdgeList of sexpr * sexpr list
  | SEdgeOp of sexpr * sexpr * op * sexpr * sexpr
  | SEdgeOpBi of sexpr * sexpr * op * sexpr * sexpr * sexpr
  | SNoexpr

type sstmt =
    SBlock of sstmt list
  | SExpr of sexpr
  | SIf of sexpr * sstmt * sstmt
  | SBfs of sexpr * sexpr * sexpr * sstmt
  | SWhile of sexpr * sstmt
  | SBinding of bind
  | SBinding_Assign of bind * sexpr
  | SReturn of sexpr

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

type sprogram = sfunc_decl list
```

```
(* Pretty-printing functions *)

let rec string_of_sexpr (t, e) =
  "(" ^ string_of_typ t ^ " : " ^ (match e with
    SLiteral(l) -> string_of_int l
  | SFLit(l) -> l
  | SBoolLit(true) -> "true"
  | SBoolLit(false) -> "false"
  | SChrLit(c) -> Char.escaped c
  | SNodeLit(_, name)  -> string_of_sexpr name
  | SGraphLit(name) -> name
  | SStrLit(str) -> str
  | SId(s) -> s
  | SNodeAttr(e1, t, e2) ->
      string_of_sexpr e1 ^ " " ^ string_of_typ t ^ " " ^  string_of_sexpr e2
  | SAttr(sx, a, e, e2) -> (match e with
                      (_, SNoexpr) | (Void, _) -> string_of_sexpr sx ^ "." ^ a
                    | _ -> string_of_sexpr sx ^ "." ^ a ^ "[" ^ string_of_sexpr e
                           ^ ", " ^ string_of_sexpr e2 ^ "]"  )
  | SBinop(e1, o, e2) ->
      string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_sexpr e2
  | SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
  | SAssign(v, e1, e2) | SAssignNode(v, e1, e2) ->
      (match e2 with
          (Void, SNoexpr) -> v ^ " = " ^ string_of_sexpr e1
        | _ -> v ^ "[" ^ string_of_sexpr e1 ^ "]" ^ " = " ^ string_of_sexpr e2
      )
  | SCall(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^ ")"
  | SAccess(id, e) -> id ^ "[" ^ string_of_sexpr e ^ "]"
  | SNoexpr -> ""
  | SEdgeOpBi(_, e1, o, e3, e4, e5) -> string_of_sexpr e1 ^ " " ^ string_of_sexpr e5 ^
string_of_op o ^ "|"
        ^ string_of_sexpr e3 ^ "| " ^ string_of_sexpr e4
  | SEdgeOp(_, e1, o, e3, e4) -> string_of_sexpr e1 ^ " " ^ string_of_op o ^ "|"
        ^ string_of_sexpr e3 ^ "| " ^ string_of_sexpr e4
  | SEdgeList(e1, e2) -> string_of_sexpr e1 ^ ": "
        ^ (List.fold_left (fun s e -> s ^ match e with
               (_, SEdgeOp(_, e2, o, e3, e4)) -> (match o with
                      Link -> (string_of_sexpr e2 ^ " " ^ string_of_op o
                       ^ string_of_sexpr e3 ^ " ")
                    | _ -> "[" ^ string_of_op o ^ " " ^ string_of_sexpr e4 ^ "] ")
             |  (_, SEdgeOpBi(_, e2, o, e3, _, e5)) ->
```

```
                      (string_of_sexpr e2 ^ " " ^ string_of_sexpr e5 ^ string_of_op o
                       ^ string_of_sexpr e3 ^ " ")
             | _ -> ""
       ) "" (List.rev e2))
       ^ match (List.hd e2) with
             (_, SEdgeOp(_, _, _, _, e4)) -> string_of_sexpr e4
           | (_, SEdgeOpBi(_, _, _, _, e4, _)) -> string_of_sexpr e4
           | _ -> ""
                       ) ^ ")"

let rec string_of_sstmt = function
    SBlock(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_sstmt stmts) ^ "}\n"
  | SExpr(expr) -> string_of_sexpr expr ^ ";\n";
  | SIf(e, s, SBlock([])) ->
      "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s
  | SIf(e, s1, s2) ->  "if (" ^ string_of_sexpr e ^ ")\n" ^
      string_of_sstmt s1 ^ "else\n" ^ string_of_sstmt s2
  | SBfs(e1, e2, e3, s) ->
      "for (" ^ string_of_sexpr e1  ^ " ; " ^ string_of_sexpr e2 ^ " ; " ^
      string_of_sexpr e3  ^ ") " ^ string_of_sstmt s
  | SWhile(e, s) -> "while (" ^ string_of_sexpr e ^ ") " ^ string_of_sstmt s
  | SBinding(t, id) -> "(" ^ string_of_typ t ^ " : " ^ id ^ ");\n"
  | SBinding_Assign((t, id), e) ->
        string_of_typ t ^ " " ^ id ^ " = " ^ string_of_sexpr e ^ ";\n"
  | SReturn(e) -> "return " ^ string_of_sexpr e ^ ";\n"

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

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

19.6 codegen.ml (Adam, Jack, James, Shvetank)

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

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

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

Detailed documentation on the OCaml LLVM library:

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

*)

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

module StringMap = Map.Make(String)

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

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

  (* Get types from the context *)
  let i32_t      = L.i32_type    context
  and float_t    = L.double_type context
  and i8_t       = L.i8_type     context
  and i1_t       = L.i1_type     context
  and i64_t      = L.i64_type    context
  and void_t     = L.void_type   context
  in

  (* Graph Types *)
  let node_t      = L.named_struct_type context "node_t" in
  L.struct_set_body node_t [| i32_t; L.pointer_type i8_t; i8_t; i32_t |] true;

  let edge_t      = L.named_struct_type context "edge_t" in
  L.struct_set_body edge_t [| L.pointer_type node_t; L.pointer_type node_t; i32_t; i32_t  |]
true;

  let edge_list_t =  L.named_struct_type context "edge_list_t" in
  L.struct_set_body edge_list_t [| L.pointer_type edge_t; L.pointer_type edge_list_t |]
```

```
true;

  let graph_t     = L.named_struct_type context "graph_t" in
  L.struct_set_body graph_t [| i32_t; i32_t; i32_t; L.pointer_type (L.pointer_type node_t);
                               L.pointer_type (L.pointer_type edge_list_t) |] true;

  (* Return the LLVM type for a YAGL type *)
  let rec ltype_of_typ = function
      A.Int           -> i32_t
    | A.Float         -> float_t
    | A.String        -> L.pointer_type i8_t
    | A.Char          -> i8_t
    | A.Void          -> void_t
    | A.Bool          -> i1_t
    | A.Node          -> L.pointer_type node_t
    | A.Graph         -> L.pointer_type graph_t
    | A.Edge          -> L.pointer_type edge_t
    | A.Array (t, e) -> let num =(match e with
                          Literal(l) -> l
                        | Binop(_, _, _) -> raise(Failure("TODO: Not currently
supported."))
                        | Id _  -> raise(Failure("TODO: Not currently supported."))
                        | _ -> raise(Failure("Can not declare array's length with non
integer expr type."))
                        (* This is for declaring an array; these are supported for
accessing an array though!*)
                       )
                       in L.array_type (ltype_of_typ t) num
  in

  (* Declare built-in functions *)
  let printf_t : L.lltype =
      L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
  let printf_func : L.llvalue =
      L.declare_function "printf" printf_t the_module in
  let make_graph_t : L.lltype =
          L.function_type (L.pointer_type graph_t)
          [| i32_t |] in
  let update_node_t : L.lltype =
          L.function_type (L.pointer_type node_t)
           [| (L.pointer_type node_t); (L.pointer_type i8_t) |] in
  let get_name_node_t : L.lltype =
          L.function_type ((L.pointer_type i8_t) )
           [| (L.pointer_type node_t) |] in
```

```
  let get_node_t : L.lltype =
          L.function_type ((L.pointer_type node_t) )
           [| (L.pointer_type graph_t); i32_t |] in
  let get_neighbor_t : L.lltype =
          L.function_type ((L.pointer_type node_t) )
           [| (L.pointer_type graph_t); (L.pointer_type node_t); i32_t |] in
  let get_num_neighbors_t : L.lltype =
          L.function_type (i32_t)
           [| (L.pointer_type graph_t); (L.pointer_type node_t); |] in
  let get_weight_t : L.lltype =
          L.function_type (i32_t)
           [| (L.pointer_type graph_t); (L.pointer_type node_t); (L.pointer_type node_t)|]
in
  let insert_edge_t : L.lltype =
          L.function_type (L.pointer_type graph_t)
           [| (L.pointer_type graph_t); (L.pointer_type node_t); i32_t; (L.pointer_type
node_t) |] in
  let remove_node_t : L.lltype =
          L.function_type (L.pointer_type graph_t)
           [| (L.pointer_type graph_t); (L.pointer_type node_t) |] in
  let insert_node_t : L.lltype =
          L.function_type (L.pointer_type graph_t)
          [| L.pointer_type graph_t; L.pointer_type node_t |] in
  let print_graph_t : L.lltype =
          L.function_type i32_t [| (L.pointer_type graph_t)|] in
  let sconcat_t : L.lltype =
          L.function_type (L.pointer_type i8_t)
          [| L.pointer_type i8_t; L.pointer_type i8_t |] in
  let make_graph_func : L.llvalue =
      L.declare_function "make_graph" make_graph_t the_module in
  let update_node_func : L.llvalue =
      L.declare_function "update_node" update_node_t the_module in
  let insert_edge_func : L.llvalue =
      L.declare_function "insert_edge" insert_edge_t the_module in
  let remove_node_func : L.llvalue =
      L.declare_function "remove_node" remove_node_t the_module in
  let get_node_func : L.llvalue =
      L.declare_function "get_node" get_node_t the_module in
  let get_weight_func : L.llvalue =
      L.declare_function "get_weight" get_weight_t the_module in
  let get_name_node_func : L.llvalue =
      L.declare_function "get_name_node" get_name_node_t the_module in
  let get_neighbor_func : L.llvalue =
      L.declare_function "get_neighbor" get_neighbor_t the_module in
```

```
  let get_num_neighbors_func : L.llvalue =
      L.declare_function "get_num_neighbors" get_num_neighbors_t the_module in
  let insert_node_func : L.llvalue =
      L.declare_function "insert_node" insert_node_t the_module in
  let print_graph_func : L.llvalue =
      L.declare_function "print_graph" print_graph_t the_module in
  let sconcat_func : L.llvalue =
      L.declare_function "sconcat" sconcat_t the_module in
  let strlen_t : L.lltype =
      L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
  let strlen_func : L.llvalue =
      L.declare_function "strlen" strlen_t the_module in
  let is_visited_t : L.lltype =
      L.var_arg_function_type (i1_t) [| L.pointer_type node_t |] in
  let is_visited_func : L.llvalue =
      L.declare_function "is_visited" is_visited_t the_module in
  let update_visited_t : L.lltype =
      L.var_arg_function_type (L.pointer_type node_t) [| (L.pointer_type node_t); i1_t |] in
  let update_visited_func : L.llvalue =
      L.declare_function "update_visited" update_visited_t the_module in
  let get_distance_t : L.lltype =
      L.var_arg_function_type (i32_t) [| L.pointer_type node_t |] in
  let get_distance_func : L.llvalue =
      L.declare_function "get_distance" get_distance_t the_module in
  let update_distance_t : L.lltype =
      L.var_arg_function_type (L.pointer_type node_t) [| (L.pointer_type node_t); i32_t |]
in
  let update_distance_func : L.llvalue =
      L.declare_function "update_distance" update_distance_t the_module in

  (* Graph related calls *)
  let make_node_t : L.lltype =
      L.var_arg_function_type (L.pointer_type node_t) [| L.pointer_type i8_t |] in
  let make_node_func : L.llvalue =
      L.declare_function "make_node" make_node_t the_module in
  let print_node_t : L.lltype =
      L.var_arg_function_type i32_t [| L.pointer_type node_t |] in
  let print_node_func : L.llvalue =
      L.declare_function "print_node" print_node_t the_module in


  (* Define each function (arguments and return type) so we can
     call it even before we've created its body *)
  let function_decls : (L.llvalue * sfunc_decl) StringMap.t =
```

```
  let function_decl m fdecl =
    let name = fdecl.sfname
    and formal_types =
    Array.of_list (List.map (fun (t,_) -> ltype_of_typ t) fdecl.sformals)
    in let ftype = L.function_type (ltype_of_typ fdecl.styp) formal_types in
    StringMap.add name (L.define_function name ftype the_module, fdecl) m in
  List.fold_left function_decl StringMap.empty functions in

(* Fill in the body of the given function *)
let build_function_body fdecl =
  let (the_function, _) = StringMap.find fdecl.sfname function_decls in
  let builder = L.builder_at_end context (L.entry_block the_function) in

  let int_format_str = L.build_global_stringptr "%d\n" "fmt" builder
  and float_format_str = L.build_global_stringptr "%g\n" "fmt" builder
  and string_format_str = L.build_global_stringptr "%s\n" "fmt" builder
  and char_format_str = L.build_global_stringptr "%c\n" "fmt" builder in


  (* Construct the function's "locals": formal arguments and locally
     declared variables.  Allocate each on the stack, initialize their
     value, if appropriate, and remember their values in the "locals" map *)
  let local_vars =
    let add_formal m (t, n) p =
      L.set_value_name n p;
let local = L.build_alloca (ltype_of_typ t) n builder in
      ignore (L.build_store p local builder);
StringMap.add n local m

    (* Allocate space for any locally declared variables and add the
     * resulting registers to our map *)
    and add_local m (t, n) =
let local_var = L.build_alloca (ltype_of_typ t) n builder
in StringMap.add n local_var m
    in

    let formals = List.fold_left2 add_formal StringMap.empty fdecl.sformals
        (Array.to_list (L.params the_function)) in
    List.fold_left add_local formals (List.fold_left

    (fun bind_list stmt ->
      match stmt with
        SBinding b -> b :: bind_list
      | SBinding_Assign (b, _) -> b :: bind_list
```

```
        | _ -> bind_list
      ) [] fdecl.sbody)

    in

    (* Return the value for a variable or formal argument.
       Check local names first, then global names *)
    let outermost_func_scope  = [local_vars]
    in

    let rec lookup n symbol_tables_list = match symbol_tables_list with
       last_map :: []        -> (try StringMap.find n last_map with Not_found -> raise
(Failure ("Should have been caught in semant.")))
     | head_map :: tail_maps -> (try StringMap.find n head_map with Not_found -> lookup n
tail_maps)
     | []                    -> raise(Failure("Internal Error: Symbol table not built."))
    in

    (* Construct code for an expression; return its value *)
    let rec expr builder s_table ((_, e) : sexpr) = match e with
      SLiteral i  -> L.const_int i32_t i
     | SFLit f -> L.const_float_of_string float_t f
     | SEdgeList (e1, e2) ->
                    let _ = List.map (fun ele ->
                      expr builder s_table ele) (List.rev e2) in
                    expr builder s_table e1
     | SEdgeOpBi (e1, e2, _, e3, e4, e5) ->
         let e1' = expr builder s_table e1
         and e2' = expr builder s_table e2
         and e3' = expr builder s_table e3
         and e4' = expr builder s_table e4
         and e5' = expr builder s_table e5 in
                      ignore (L.build_call
                      insert_edge_func [| e1'; e2'; e3'; e4' |] "insert_edge" builder);
                      L.build_call
                      insert_edge_func [| e1'; e4'; e5'; e2' |] "insert_edge" builder
     | SEdgeOp (e1, e2, op, e3, e4) ->
         let e1' = expr builder s_table e1
         and e2' = expr builder s_table e2
         and e3' = expr builder s_table e3
         and e4' = expr builder s_table e4 in
         (match op with
           A.Link      -> L.build_call
                      insert_edge_func [| e1'; e2'; e3'; e4' |] "insert_edge" builder
```

```
          | A.Sub  -> L.build_call
                      remove_node_func [| e1'; e4' |] "remove_node" builder
          | A.RevLink   -> L.build_call
                           insert_edge_func [| e1'; e4'; e3'; e2' |] "insert_edge" builder
          | A.BiLink    -> ignore (L.build_call
                           insert_edge_func [| e1'; e2'; e3'; e4' |] "insert_edge" builder);
                               L.build_call
                           insert_edge_func [| e1'; e4'; e3'; e2' |] "insert_edge" builder
          | A.Add       -> L.build_call
                           insert_node_func [| e1'; e4' |] "insert_node" builder
          | _ -> raise (Failure "This edge op is not implemented.")
          )
      | SId s   -> L.build_load (lookup s s_table) s builder
      | SAttr ((String, sId), "length", _, _) ->
          L.build_call strlen_func [| (expr builder s_table (String, sId)) |] "strlen"
builder
      | SAttr ((Node, sId), "visited", _, _) ->
          L.build_call is_visited_func [| (expr builder s_table (Node, sId)) |]
"is_visited" builder
      | SAttr ((Node, sId), "curr_dist", _, _) ->
          L.build_call get_distance_func [| (expr builder s_table (Node, sId)) |]
"get_distance" builder
      | SAttr ((Graph, sId), attr, e, e2) -> (match attr with
          "node"              -> L.build_call get_node_func [| (expr builder s_table
(Graph, sId)) ; expr builder s_table e |] "get_node"
          | "num_nodes"         ->      let e' = expr builder s_table (Node, sId) in
                                        ignore (L.set_alignment 4 e');
                                        let ptr = L.build_struct_gep e' 1 "get_num_nodes"
builder in
                                        L.build_load ptr "num_nodes_ptr"
          | "neighbor"          -> L.build_call get_neighbor_func [| (expr builder s_table
(Graph, sId)) ; expr builder s_table e; expr builder s_table e2 |] "get_neighbor"
          | "num_neighbors"     -> L.build_call get_num_neighbors_func [| (expr builder
s_table (Graph, sId)) ; expr builder s_table e |] "get_num_neighbors"
          | "weight"            -> L.build_call get_weight_func [| (expr builder s_table
(Graph, sId)) ; expr builder s_table e; expr builder s_table e2 |] "get_weight"
          | _ -> raise (Failure "unsupported attribute type")) builder
      | SAttr ((Node, sId), "name", _, _) ->
            let e' = expr builder s_table (Node, sId) in
               ignore (L.set_alignment 8 e');
            let ptr = L.build_struct_gep e' 1  "get_name" builder in
            let x = L.build_load ptr "get_name_load" builder in
            ignore (L.set_alignment 8 x); ignore(x);
            L.build_call get_name_node_func [| (expr builder s_table (Node, sId)) |]
```

```
"get_name_node" builder
      | SAttr (_) ->
          raise (Failure "unsupported attribute type")
      | SNodeLit (_, nodeName) ->
                    L.build_call make_node_func [| (expr builder s_table nodeName) |]
                    "make_node" builder
      | SBinop ((A.Graph, _ ) as e1, op, e2) ->
        let e1' = expr builder s_table e1
        and e2' = expr builder s_table e2 in
          (match e2 with
                (A.Node, _) ->
                      (match op with
                                  A.Add -> L.build_call insert_node_func [| e1'; e2' |]
"insert_node" builder
                                | A.Sub -> L.build_call remove_node_func [| e1'; e2' |]
"remove_node" builder
                                | _ -> raise (Failure "Internal error: Semant should've
caught")
                       )
              | _ -> raise (Failure "Internal error: Semant should've caught")
          )
      | SBinop ((A.Float,_ ) as e1, op, e2) ->
        let e1' = expr builder s_table e1
        and e2' = expr builder s_table e2 in
        (match op with
          A.Add     -> L.build_fadd
        | A.Sub     -> L.build_fsub
        | A.Mult    -> L.build_fmul
        | A.Div     -> L.build_fdiv
        | A.Equal   -> L.build_fcmp L.Fcmp.Oeq
        | A.Less    -> L.build_fcmp L.Fcmp.Olt
        | A.Greater -> L.build_fcmp L.Fcmp.Ogt
        | A.And | A.Or ->
            raise (Failure "internal error: semant should have rejected and/or on float")
          | _ -> raise (Failure "This float binop is not implemented")
        ) e1' e2' "tmp" builder
      | SBinop (((A.String,_ )) as e, op, e2) ->
          if op == A.Add then
                L.build_call sconcat_func [| (expr builder s_table e); (expr builder s_table
e2)  |]
              "sconcat" builder
          else
                raise (Failure "internal error: can only concatenate (+) strings")
      | SBinop (e1, op, e2) ->
```

```
        let e1' = expr builder s_table e1
        and e2' = expr builder s_table e2 in
        (match op with
          A.Add     -> L.build_add
        | A.Sub     -> L.build_sub
        | A.Mult    -> L.build_mul
          | A.Div     -> L.build_sdiv
        | A.And     -> L.build_and
        | A.Or      -> L.build_or
        | A.Equal   -> L.build_icmp L.Icmp.Eq
        | A.Less    -> L.build_icmp L.Icmp.Slt
        | A.Greater -> L.build_icmp L.Icmp.Sgt
          | _         -> raise (Failure "This binop is not implemented")
        ) e1' e2' "tmp" builder
      | SNodeAttr (e1, t, e2) ->
          let e1' = expr builder s_table e1
        and e2' = expr builder s_table e2 in
          (match t with
            A.Bool ->
                L.build_call update_visited_func [| e1'; e2'|] "update_visited"
          | A.Int  -> L.build_call update_distance_func [| e1'; e2'|] "update_distance"
          | _         -> raise (Failure "This NodeAttr is not implemented")
          ) builder
      | SStrLit  s  -> L.build_global_stringptr s "fmt" builder
      | SChrLit  c  -> L.const_int i8_t (Char.code c)
      | SBoolLit b  -> L.const_int i1_t (if b then 1 else 0)
      | SGraphLit _ ->
                L.build_call make_graph_func [| L.const_int i32_t 1 |]
                "make_graph" builder
      | SAssignNode (s, e1, e2) -> (match e2 with
                              (_, SNoexpr) -> (let e' = expr builder s_table e1 in
                              ignore (L.build_call update_node_func [| L.build_load (lookup
s s_table) "ptr" builder; e'  |]
                              "update_node" builder); e')
                              | _ -> let e' = expr builder s_table e2 in
                                     let index = (match e1 with (* expr builder e in *)
                                        (Int, _)          -> expr builder s_table e1
                                     (*| (Int, SLiteral l) -> L.const_int i64_t l May want
to keep? *)
                                       | _                 -> raise(Failure("Semant.ml
should have caught."))
                                     ) in
                                     let indices =
                                       (Array.of_list [L.const_int i64_t 0; index]) in
```

```
                                    let ptr =
                                      L.build_in_bounds_gep (lookup s s_table) indices
(s^"_ptr_") builder
                                    in L.build_store e' ptr builder
                                )
      | SAssign (s, e1, e2) -> (match e2 with
                                (_, SNoexpr) -> (let e' = expr builder s_table e1 in
                                        ignore(L.build_store e' (lookup s s_table) builder);
e')
                                | _ -> let e' = expr builder s_table e2 in
                                    let index = (match e1 with (* expr builder e in *)
                                       (Int, _)          -> expr builder s_table e1
                                     (*| (Int, SLiteral l) -> L.const_int i64_t l May want
to keep? *)
                                      | _                -> raise(Failure("Semant.ml
should have caught."))
                                    ) in
                                    let indices =
                                      (Array.of_list [L.const_int i64_t 0; index]) in
                                    let ptr =
                                      L.build_in_bounds_gep (lookup s s_table) indices
(s^"_ptr_") builder
                                    in L.build_store e' ptr builder
                                )
      | SCall ("print", [x]) -> (
                        match x with
                            (Node, _) -> (L.build_call print_node_func [| expr builder s_table
x |] "print_node" builder)
                          | (Graph, _) -> L.build_call print_graph_func [| expr builder
s_table x |] "print_graph" builder
                          | (Int, _) | (Bool, _) -> L.build_call printf_func [| int_format_str
; (expr builder s_table x) |] "printf" builder
                          | (Char, _) -> L.build_call printf_func [| char_format_str ; (expr
builder s_table x) |] "printf" builder
                          | (Float, _) -> L.build_call printf_func [| float_format_str ; (expr
builder s_table x) |] "printf" builder
                          | (String, _) -> L.build_call printf_func [| string_format_str ;
(expr builder s_table x) |] "printf" builder
                          | _ -> raise (Failure("Not implemented print type.")))
      | SCall ("printNode", [n]) ->
    L.build_call print_node_func [| expr builder s_table n |] "print_node" builder
      | SCall ("printGraph", [g]) ->
        L.build_call print_graph_func [| expr builder s_table g |] "print_graph" builder
      | SCall ("printInt", [e]) | SCall ("printBool", [e]) ->
```

```
        L.build_call printf_func [| int_format_str ; (expr builder s_table e) |]
          "printf" builder
      | SCall ("printChar", [e])  ->
        L.build_call printf_func [| char_format_str ; (expr builder s_table e) |]
          "printf" builder
      | SCall ("printFloat", [e]) ->
    L.build_call printf_func [| float_format_str ; (expr builder s_table e) |]
      "printf" builder
      | SCall ("printString", [e]) ->
        L.build_call printf_func [| string_format_str ; (expr builder s_table e) |]
          "printf" builder
      | SCall ("printf", [e]) ->
    L.build_call printf_func [| float_format_str ; (expr builder s_table e) |]
      "printf" builder
      | SCall (f, args) ->
         let (fdef, fdecl) = StringMap.find f function_decls in
       let llargs = List.rev (List.map (expr builder s_table) (List.rev args)) in
       let result = (match fdecl.styp with
                        A.Void -> ""
                      | _ -> f ^ "_result") in
        L.build_call fdef (Array.of_list llargs) result builder
      | SAccess (s, e) -> let index = (match e with (* expr builder e in *)
                             (Int, _)          -> expr builder s_table e
                           (*| (Int, SLiteral l) -> L.const_int i64_t l  We might want to
check this? *)
                            | _                -> raise(Failure("This should have been
caught by semant.ml"))
                           ) in
                           let indices =
                             (Array.of_list [L.const_int i64_t 0; index]) in
                           let ptr =
                             L.build_in_bounds_gep (lookup s s_table) indices (s^"_ptr_")
builder
                           in L.build_load ptr (s^"_elem_") builder
      | SUnop(op, ((t, _) as e)) ->
          let e' = expr builder s_table e in
        (match op with
          A.Neg when t = A.Float -> L.build_fneg
        | A.Neg                  -> L.build_neg
          | A.Not                  -> L.build_not) e' "tmp" builder
      | _ -> raise (Failure("Unhandled case: unimplemented"))
    in

    (* LLVM insists each basic block end with exactly one "terminator"
```

```
      instruction that transfers control.  This function runs "instr builder"
      if the current block does not already have a terminator.  Used,
      e.g., to handle the "fall off the end of the function" case. *)
    let add_terminal builder instr =
      match L.block_terminator (L.insertion_block builder) with
      Some _ -> ()
      | None -> ignore (instr builder) in

    (* Build the code for the given statement; return the builder for
       the statement's successor (i.e., the next instruction will be built
       after the one generated by this call) *)

    let rec stmt s_table builder s1 = match s1 with
      SBlock sl -> let add_local m (t, n) =
                     let local_var = if t != Node then
                                             L.build_alloca (ltype_of_typ t) n builder
                                     else
                                             let x = L.build_alloca (ltype_of_typ t) n
builder in
                                             L.set_alignment 8 x;
                                             x
                     in StringMap.add n local_var m
                   in
                   let updated_table =  (List.fold_left add_local StringMap.empty
(List.fold_left
                                           (fun bind_list stmt ->
                                             match stmt with
                                               SBinding b -> b :: bind_list
                                             | SBinding_Assign (b, _) -> b :: bind_list
                                             | _ -> bind_list
                                           ) [] sl)
                                         ) :: s_table
                                         in List.fold_left (stmt updated_table) builder sl
      | SExpr e -> ignore(expr builder s_table e); builder
      | SBinding (_,_) -> builder;
      | SBinding_Assign ((_, _), e) -> ignore (expr builder s_table e); builder
      | SReturn e -> ignore(match fdecl.styp with
                (* Special "return nothing" instr *)
                A.Void -> L.build_ret_void builder
                (* Build return statement *)
                | _ -> L.build_ret (expr builder s_table e) builder );
            builder
      | SIf (predicate, then_stmt, else_stmt) ->
         let bool_val = expr builder s_table predicate in
```

```
      let merge_bb = L.append_block context "merge" the_function in
        let build_br_merge = L.build_br merge_bb in (* partial function *)

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

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

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

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

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

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


     | _ -> raise (Failure("Only support expression statements currently."))

  in

  (* Build the code for each statement in the function *)
  let builder = stmt outermost_func_scope builder (SBlock fdecl.sbody) in

  (* Add a return if the last block falls off the end *)
  add_terminal builder (match fdecl.styp with
      A.Void -> L.build_ret_void
    | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
in

List.iter build_function_body functions;
the_module
```

19.7 yagl.ml (with addition of preprocessor) (Shvetank)

```
(*
Implementation for language
Top-level of the YAGL compiler: scan & parse the input,
   check the resulting AST and generate an SAST from it, generate LLVM IR,
   and dump the module

Minor helper functions of the large preprocessing code for file reading were adapted from
Stack Overflow.
*)


type action = Ast | Compile  | Sast | LLVM_IR

let () =
  let action = ref Compile in
  let set_action a () = action := a in
  let speclist = [
    ("-a", Arg.Unit (set_action Ast), "Print the AST");
    ("-s", Arg.Unit (set_action Sast), "Print the SAST");
    ("-l", Arg.Unit (set_action LLVM_IR), "Print the generated LLVM IR");
    ("-c", Arg.Unit (set_action Compile),
      "Check and print the generated LLVM IR (default)");
  ] in

  (* Preprocessing of file for imports *)
  let usage_msg = "usage: ./yagl.native [-a|-s|-l|-c] [file.ygl]" in
  let file_name = ref "" in
  let () = Arg.parse speclist (fun filename -> file_name := filename) usage_msg in
  let read_file =
  let lines = ref [] in
  let chan = open_in !file_name in
  try
    while true; do
      lines := input_line chan :: !lines
      done; !lines
  with End_of_file ->
    close_in chan;
    List.rev !lines
  in
  let file_lines = read_file in
  let contains_import l1 =
```

```
  try
    let len = String.length "import" in
    for i = 0 to String.length l1 - len do
      if String.sub l1 i len = "import" then raise Exit
    done;
    false
  with Exit -> true
  in

  let read_import line =
     let imported_file = List.nth (String.split_on_char ' ' line) 1 in
     let read_whole_file =
         let ch = open_in imported_file in
         let s = really_input_string ch (in_channel_length ch) in
             close_in ch;
             s
     in read_whole_file
  in
  let import_file prev_lines curr_line = if contains_import curr_line then (read_import
curr_line)::prev_lines else curr_line::prev_lines in
  let new_file_lines = (List.rev (List.fold_left import_file [] file_lines)) in
  let rec print_new_file l oc = match l with
     [] -> ()
     | h::t -> ignore(Printf.fprintf oc "%s\n" h); print_new_file t oc
  in
  let new_file_name = (List.nth (String.split_on_char '.' !file_name) 0) ^
"_preprocessed.ygl" in
  let new_file_channel = open_out new_file_name
  in print_new_file new_file_lines new_file_channel; close_out new_file_channel;


  (* Execute with preprocessed file now and remove it at the end*)
  let channel = ref stdin in  channel := open_in new_file_name;
  let lexbuf = Lexing.from_channel !channel in
  let ast = Yaglparse.program Scanner.token lexbuf in
  match !action with
    Ast -> print_string (Ast.string_of_program ast)
    | _ -> let sast = Semant.check ast in

    match !action with
      Ast     -> ()
    | Sast    -> print_string (Sast.string_of_sprogram sast)
    | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate sast))
    | Compile -> let m = Codegen.translate sast in
```

```
      Llvm_analysis.assert_valid_module m;
      print_string (Llvm.string_of_llmodule m)
  ;
  Sys.remove new_file_name
```

19.8 Standard Library Code (in YAGL) (James)

19.8.0.1 stdgraph.ygl

```
/* YAGL's standard graph */
Graph copy_graph_lib(Graph g) {

      Graph g2;

      int nodes = g.num_nodes;
      int curr = 0;
      while (curr < nodes) {
            Node current = g.node[curr];
            g2: + current;
            curr = curr + 1;
      }

      curr = 0;
      while (curr < nodes) {

            Node current = g.node[curr];
            int neighs = g.num_neighbors[current];
            int on = 0;
            while (on < neighs) {
                  Node to;
                  to  = g.neighbor[current, on];
                  int val = g.weight[current, to];
                  g2: current ->val to;
                  on = on + 1;
            }

            curr = curr + 1;
      }

      return g2;

}

Graph reverse_graph_lib(Graph g) {
```

```
	Graph g2;

	int nodes = g.num_nodes;
	int curr = 0;
	while (curr < nodes) {
		Node current = g.node[curr];
		g2: + current;
		curr = curr + 1;
	}

	curr = 0;
	while (curr < nodes) {

		Node current = g.node[curr];
		int neighs = g.num_neighbors[current];
		int on = 0;
		while (on < neighs) {
			Node to;
			to  = g.neighbor[current, on];
			int val = g.weight[current, to];
			g2: to ->val current; /* Reversed */
			on = on + 1;
		}

		curr = curr + 1;
	}
	return g2;
}

void print_graph_lib(Graph g) {
	int nodes = g.num_nodes;
	int curr = 0;
	String nodes_s = "Nodes: ";
	String edges = "Edges: ";
	while (curr < nodes) {
		Node current = g.node[curr];
		nodes_s = nodes_s + current.name + " ";
		curr = curr + 1;
		int neighs = g.num_neighbors[current];
		int on = 0;
		while (on < neighs) {
			Node ne = g.neighbor[current, on];
			String edge = current.name + " -> ";
			edges = edges + edge + ne.name;
```

```
                edges = edges + ", ";
                on = on + 1;
            }
        }
        print(nodes_s);
        print(edges);
}


Node[1000] get_node_array(Graph g) {
        Node[1000] nn;
        int size = g.num_nodes;
        int curr = 0;
        while (curr < size) {
            nn[curr] = g.node[curr];
            curr = curr + 1;
        }
        return nn;
}
```

*Please note that the above YAGL code block may include incomplete or inaccurate code highlighting.

19.8.0.2 stdalgo.ygl

```
void reset(Graph g) {
    int size = g.num_nodes;
    int curr = 0;

    while (curr < size) {
      Node current;
      current = g.node[curr];
      current.visited = false;
      current.curr_dist = 0;
      curr = curr + 1;
    }
}


/* YAGL's Standard Graph Library */
void dfs_helper(Graph g, Node vertex, int depth){
   if (vertex.visited == true) {
       return;
   }
   if (vertex.curr_dist > depth) {
       return;
```

```
   }
   vertex.visited = true;

   /* Do whatever you want to the vertex */
   if (vertex.curr_dist == 0) {
   } else {
      print(vertex);
   }

   int size = g.num_neighbors[vertex];
   int curr = 0;

   while (curr < size) {
      Node current;
      current = g.neighbor[vertex, curr];
      if (current.curr_dist == 0) {
          current.curr_dist = vertex.curr_dist + 1;
      }
      curr = curr + 1;
   }
   curr = 0;
   while (curr < size) {
      Node current;
      current = g.neighbor[vertex, curr];
      /*print(g.weight[current, vertex]);*/
      dfs_helper(g, current, depth);
      curr = curr + 1;
   }
}

void dfs(Graph g, Node vertex, int depth) {
      reset(g);
      dfs_helper(g, vertex, depth);
}


Node get_first_node_at_depth(Graph g, Node vertex, Node b, int depth) {
      reset(g);
      return get_first_node_at_depth_helper(g, vertex, b, depth);
}

Node get_first_node_at_depth_helper(Graph g, Node vertex, Node break, int depth){
   if (vertex.visited == true) {
       return break;
```

```
   }

   if (depth == vertex.curr_dist) {
       return vertex;
   }
   vertex.visited = true;

   int size = g.num_neighbors[vertex];
   int curr = 0;

   while (curr < size) {
      Node current;
      current = g.neighbor[vertex, curr];
      if (current.curr_dist == 0) {
          current.curr_dist = vertex.curr_dist + 1;
      }
      curr = curr + 1;
   }
   curr = 0;
   Node new; /* just a place holder */
   while (curr < size) {
      Node current;
      current = g.neighbor[vertex, curr];
      new = get_first_node_at_depth_helper(g, current, break, depth);
      if (new == break) {
      } else {
        return new;
      }
      curr = curr + 1;
   }
   return new;
}
```

*Please note that the above YAGL code block may include incomplete or inaccurate code highlighting.

19.9 Built-In Functions Code (in C) (James)

19.9.0.1 stdlib.c

```
/*
 *  Standard library functions in C for YAGL language
 */

#include <stdio.h>
#include <fcntl.h>    /* For O_RDWR */
```

```c
#include <unistd.h>   /* For open(), creat() */
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>

int check_null(void *x) {
      int nullfd = open("/dev/random", O_WRONLY);
      if (!(write(nullfd, x, 1) < 0))
            return 1;
      return 0;
}

char *sconcat(char *s1, char *s2) {
      char *s3 = malloc(strlen(s1) + strlen(s2) + 1);
      strcpy(s3, s1);
      strcat(s3, s2);
      return s3;
}


struct node {
      int id;  // for hash table
      char* name;
      bool visited;
      int curr_dist;
};

struct graph {
      int n_size;
      int n_pos;
      int e_pos;
      struct node **nodes;
      struct edge_list **edges;
};

struct edge {
      struct node *from_node;
      struct node *to_node;
      int val;
      int deleted; // 0 is no 1 is yes
};

struct edge_list {
      struct edge *edge;
```

```
	struct edge_list *next_edge;
};

void copy_graph(struct graph *, struct graph *);
void insert_node(struct graph *, struct node*);
struct edge_list *new_edge_list(struct node *, struct node *, int);
void insert_into_edge_list(struct edge_list *, struct edge_list *);
int g_contain_n(struct graph *, struct node *);
char *node_to_string(struct node *);
char *edge_to_string(struct edge *);
void print_graph(struct graph *);
struct node *update_node(struct node *, char *);

struct node *update_node(struct node *n, char *name) {
	char *node_name = malloc(strlen(name) + 1);
	strcpy(node_name, name);

	char *old_name = n->name;
	n->name = node_name;
	return n;
}

/* For Jack's testing */
struct edge *make_edge(struct node *from, struct node *to, int v) {

	struct edge *edge = malloc(sizeof(struct edge));
	edge->from_node = from;
	edge->to_node = to;
	edge->val = v;
	edge->deleted = 0;
	return edge;
}

void insert_edge(struct graph *g, struct node *from, int v, struct node *to) {
	if (!g_contain_n(g, from) || !g_contain_n(g, to)) {
		printf("Nodes don't exist in given graph.\n");
		return;
	}

	// first find where to insert
	struct edge_list *list;
	int pos = g->e_pos;
	int from_id = from->id;
	for (int e = 0; e < g->e_pos; e++) {
```

```
            struct edge_list *at_e = g->edges[e];
            if (at_e != NULL) {
                  struct edge *edge = at_e->edge;
                  if (edge && edge->from_node && edge->from_node->id == from_id) {
                        pos = e;
                        edge->deleted = 0; // mark undeleted
                  }
            }
      }

      // Next: insert
      struct edge_list *holder = NULL;
      if (pos == g->e_pos) {
            g->e_pos ++;
            holder = new_edge_list(from, to, v);
            g->edges[pos] = holder;
            return;
      }
      holder = g->edges[pos];
      struct edge_list *new_el = new_edge_list(from, to, v);
      insert_into_edge_list(holder, new_el);
}

void insert_into_edge_list(struct edge_list *e, struct edge_list *new) {
      struct edge_list *curr = e;
      if (curr->edge->to_node->id == new->edge->to_node->id) {
            // update cost
            curr->edge->val = new->edge->val;
            return;
      }
      while (curr->next_edge) {
            curr = curr->next_edge;
            if (curr->edge->to_node->id == new->edge->to_node->id) {
                  // update cost
                  curr->edge->val = new->edge->val;
                  return;
            }
      }
      curr->next_edge = new;
}

struct edge_list *new_edge_list(struct node *from, struct node *to, int v) {
      struct edge_list *e = malloc(sizeof(struct edge_list));
      struct edge *edge = malloc(sizeof(struct edge));
```

```
	edge->from_node = from;
	edge->to_node = to;
	edge->val = v;
	e->edge = edge;
	e->next_edge = NULL;
	return e;
}
struct graph *make_graph(int n_size) {
	struct graph *g = malloc(sizeof(struct graph));
	g->n_size = n_size;
	g->n_pos = 0;
	g->e_pos = 0;
	g->nodes = (struct node **) malloc(sizeof(struct node *) * g->n_size);

	struct edge_list **edges = malloc(g->n_size * sizeof(struct edge_list *));
	g->edges = edges;
	return g;
}

void insert_node(struct graph *g, struct node *n) {
	if (g_contain_n(g, n)) {
		printf("Already inserted into graph.\n");
		return;
	}
	if (g->n_pos < g->n_size) {
		struct node **nodes = g->nodes;
		nodes[g->n_pos++] = n;
	} else {
		struct graph *g_new = make_graph(g->n_size*2); // double space
		copy_graph(g, g_new); // copy values of g to g_new
		memcpy(g, g_new, sizeof(*g)); // copy g_new to g in memory
		free(g_new); // clean up space
		insert_node(g, n); // insert new node
	}

}
char *node_to_string(struct node *n) {
	int name_length = strlen(n->name);
	int additional_space_safe = 30;
	char *s = malloc(additional_space_safe
			+ name_length + 1);
	sprintf(s, "(%d) : %s", n->id, n->name);
	return s;
}
```

```
char *edge_to_string(struct edge *e) {
      char *to = node_to_string(e->to_node);
      char *from = node_to_string(e->from_node);
      int val = e->val;
      int name_length = strlen(to) + strlen(from);
      int additional_space_safe = 50;
      char *s = malloc(additional_space_safe
                  + name_length + 1);
      sprintf(s, "[%s] ---(%d)---> [%s]", from, val, to);
      return s;
}
void remove_edge(struct graph *g, struct node *from, struct node *to) {
      if (!from || !to || !g)
            return;
      for (int i = 0; i < g->e_pos; i++) {
            struct edge_list *e = g->edges[i];
            struct edge_list *prev = 0;
            struct edge_list **home = &g->edges[i];
            while (e && e->edge) {
                  struct edge *edge = e->edge;
                  struct edge_list *next = e->next_edge;
                  if (edge && edge->to_node && edge->from_node &&
                      edge->to_node->id == to->id && edge->from_node->id == from->id) {
                        edge->deleted = 1;
                        /*if (prev) {
                              prev->next_edge = next;
                        } else if (next) {
                              g->edges[i] = next;
                        } else {
                              if (i != g->e_pos - 1)
                                    g->edges[i] = g->edges[i+1];
                              else
                                    g->edges[i] = 0;
                              g->e_pos--;
                        }
                        free(e);
                        return;
                  }*/

                  }
                  prev = e;
                  e = next;
            }
      }
```

```
}
void remove_node(struct graph *g, struct node *n) {
	if (!g_contain_n(g, n)) {
		printf("Graph does not contain node %s\n", node_to_string(n));
		return;
	}

	// First remove all edges that contain that node
	for (int i = 0; i < g->e_pos; i++) {
		struct edge_list *e = g->edges[i];
		struct edge_list *prev = 0;
		while (e != NULL && e->edge != NULL) {
			struct edge *edge = e->edge;
			struct edge_list *next = e->next_edge;
			if (edge->to_node->id == n->id || edge->from_node->id == n->id) {
				remove_edge(g, edge->from_node, edge->to_node);
			}
			prev = e;
			e = next;
		}
	}

	// Second remove node
	int c = 0;
	for (c ; c < g->n_pos; c++) {
		struct node *curr = g->nodes[c];
		if (curr->id == n->id)
			break;
	}

	while (c + 1 < g->n_pos) {
		memcpy(g->nodes + c, g->nodes + c+1, sizeof(g->nodes[c+1]));
		c++;
	}
	struct node *curr = g->nodes[g->n_pos - 1];
	curr = NULL;
	g->n_pos--;
}

void copy_graph(struct graph *g, struct graph *g_new) {
	for (int i = 0; i < g->n_size; i++) {
		insert_node(g_new, g->nodes[i]);
	}
	for (int n = 0; n < g->e_pos; n++) {
```

```
            struct edge_list *e = g->edges[n];
            while (e != NULL && e->edge != NULL) {
                  struct edge *edge = e->edge;
                  e = e->next_edge;
                  insert_edge(g_new, edge->from_node, edge->val, edge->to_node);
            }
      }
}
int g_contain_n(struct graph *g, struct node *n) {
      int on = 0;
      while (on < g->n_pos) {
            struct node *x = g->nodes[on++];
            if (x->id == n->id)
                  return 1;
      }
      return 0;
}

struct edge *g_contain_e(struct graph *g, struct edge *ed) {
      for (int n = 0; n < g->e_pos; n++) {
            struct edge_list *e = g->edges[n];
            while (e != NULL && e->edge != NULL) {
                  struct edge *edge = e->edge;
                  if (edge == ed)
                        return edge;
                  e = e->next_edge;
            }
      }
      return NULL;
}

static int id = 0;
struct node *make_node(char *name) {

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

      char *node_name = malloc(strlen(name) + 1);
      strcpy(node_name, name);

      n->id = id++;
      n->name = node_name;
      n->visited = false;
      n->curr_dist = 0;
      return n;
```

```
}

bool is_visited(struct node *n) {
      return n->visited;
}

void update_visited(struct node *n, bool b) {
      n->visited = b;
}

int get_distance(struct node *n) {
      return n->curr_dist;
}

void update_distance(struct node *n, int i) {
      n->curr_dist = i;
}

char *update_node_name(struct node *n, char *new_name) {

      free(n->name);

      char *node_name = malloc(strlen(new_name) + 1);
      strcpy(node_name, new_name);

      n->id = id++;
      n->name = node_name;

      return n->name;
}

void print_node(struct node *n) {
      printf("%s\n", n->name);
}

int get_graph_size(struct graph *g) {
      return g->n_pos;
}

char *get_name_node(struct node *n) {
      return n->name;
}

struct node *get_neighbor(struct graph *g, struct node *n, int pos) {
```

```
	int on = 0;
	for (int nn = 0; nn < g->e_pos; nn++) {
		struct edge_list *e = g->edges[nn];
		if (e->edge->from_node->id == n->id) {

			int nullfd = open("/dev/random", O_WRONLY);
			while (!(write(nullfd, e, sizeof(e)) < 0)) {
				struct edge *edge = e->edge;
				if (edge->deleted == 0) {
					if (on == pos)
						return edge->to_node;
					on += 1;
				}
				e = e->next_edge;
			}

		}
	}
	// Fail case: return n
	return n;
}
int get_num_neighbors(struct graph *g, struct node *n) {
	int on = 0;
	for (int nn = 0; nn < g->e_pos; nn++) {
		struct edge_list *e = g->edges[nn];
		if (e->edge->from_node->id == n->id) {

			int nullfd = open("/dev/random", O_WRONLY);
			while (!(write(nullfd, e, sizeof(e)) < 0)) {
				struct edge *edge = e->edge;
				if (edge->deleted == 0) {
					on += 1;
				}
				e = e->next_edge;
			}

		}
	}
	// Fail case: return n
	return on;
}
struct node *get_node(struct graph *g, int pos) {
	return g->nodes[pos];
}
```

```
int get_weight(struct graph *g, struct node *n, struct node *n2) {
	for (int nn = 0; nn < g->e_pos; nn++) {
		struct edge_list *e = g->edges[nn];
		if (e->edge->from_node->id == n->id) {

			int nullfd = open("/dev/random", O_WRONLY);
			while (!(write(nullfd, e, sizeof(e)) < 0)) {
				struct edge *edge = e->edge;
				if (edge->deleted == 0 && edge->to_node->id == n2->id) {
					return edge->val;
				}
				e = e->next_edge;
			}

		}
	}
	return -100000;
}

void print_graph(struct graph *g) {
	printf("============== Graph Print ===============\n");
	printf("\tStats: \t%d\t%d\t%d\n", g->n_size, g->n_pos, g->e_pos);
	int iter = 1;
	int per_line = 3;
	printf("\nNodes:\n");
	for (int n = 0; n < g->n_pos; n++)
		printf("Node %s%s", node_to_string(g->nodes[n]),
				iter++ % per_line == 0 ? "\n":
				n == g->n_pos - 1 ? "" : " --- ");
	if (g->n_pos == 0)
		printf("There aren't any nodes in this graph.\n");
	printf("\n\n");
	printf("Edges:\n");
	int printed = 0;
	for (int n = 0; n < g->e_pos; n++) {
		struct edge_list *e = g->edges[n];
		int nullfd = open("/dev/random", O_WRONLY);
		while (!(write(nullfd, e, sizeof(e)) < 0)) {
			struct edge *edge = e->edge;
			if (edge->deleted == 0) {
				printed ++;
				printf("%s\n", edge_to_string(edge));
			}
			e = e->next_edge;
```

```
            }
      }
      if (printed == 0)
            printf("There aren't any edges in this graph.\n");

      printf("============ End Graph Print =============\n");

}
#ifdef BUILD_TEST
int main()
{
      return 0;
}
#endif
```

19.10 Demos (Adam, Jack, James, Shvetank)

19.10.0.1 demo1.ygl

```
import stdalgo.ygl
import stdgraph.ygl

/* Social Media Application */

Graph fb;         /* Facebook  */
Graph tw;         /* Twitter   */
Graph ln;         /* LinkedIn  */

/* People */
Node adam("Adam");
Node james("Jamie");
/* Typo on purpose for demonstration later */
Node jack("Jack");
Node tank("Shvetank");
Node edwards("Edwards");
Node tiffany("Tiffany");
Node buehler("Buehler");
Node michael1("Michael1");
Node michael2("Michael2");



/* Add people to Facebook's network */
```

```
fb: + adam + james + jack + tank;

/* Add edges between people to make them friends */
fb: adam <-> james <-> jack, tank <-> jack, tank <-> adam;
/* Facebook friends is a bidirectional relationship */

printString("Facebook's Network");
printGraph(fb);
print_break();




/* Add people to Twitter's network and edges between them to add followers
*/
tw: + edwards + michael1 -> edwards,
/* Twitter followers is a one directional relationship (not required to
follow back) */
    + michael2 + tank <- edwards;
/* Nodes can be in multiple graphs */

printString("Twitter's Network");
printGraph(tw);
print_break();



/* Add people to LinkedIn's graphs and edges between people to make
connections, LinkedIn connections are bidirectional */
ln: + tiffany + buehler + james + edwards + michael1 + michael2 + adam +
tank <-> james <-> adam <-> edwards <-> michael1 <-> michael2 <-> tiffany
<-> edwards,
    + jack, buehler <-> james <-> jack;

printString("LinkedIn's Network");
printGraph(ln);

/* Delete the random people from LinkedIn since they chose to delete their
accounts */
ln: - tiffany - buehler - michael2 - michael1;

printString("LinkedIn's Network - with deletion");
printGraph(ln);
print_break();
```

```
/* Update typo! fixes in all graphs */
james = "James";
printString("LinkedIn's Network with James fixed");
printGraph(ln);
print_break();

print("Friends and Friends of Friends of James");
/* Find friends of friends */
dfs(fb, james, 2);           /* Should find tank */
print('\n');
print("Facebook's Network, for reference:");
print(fb);




void print_break() {
      print("");
      print("");
}
```

*Please note that the above YAGL code block may include incomplete or inaccurate code highlighting.

19.10.0.2 demo2.ygl

```
import stdgraph.ygl
import stdalgo.ygl

Graph hello; /* declare graph */

String hello_world = ""; /* declare string holder */

/* declare nodes */
Node break("");
Node h("h");
Node e("e");
Node l("l");
Node l2("l");
Node o("o");
Node space(" ");
Node w("w");
Node o2("o");
Node r("r");
```

```
Node l3("l");
Node d("d");
Node excl("!");

/* populate graph */
hello: + h + e + l + l2 + o + space + w + o2 + r + l3 + d + excl,
       h -> e -> l -> l2 -> o -> space -> w -> o2 -> r -> l3 -> d -> excl;

/* Generate array of nodes based on depth */
int num_nodes = hello.num_nodes;
Node[11] nodes;
int on = 0;
while (on < num_nodes + 1) {
      nodes[on] = get_first_node_at_depth(hello, h, break, on);
      on = on + 1;
}

/* Populate hello_world string from nodes */
on = 0;
while (on < num_nodes ) {
      Node curr = nodes[on];
      hello_world = hello_world + curr.name;
        on = on + 1;
}

/* Done! Hello world! */
print(hello_world);
```

*Please note that the above YAGL code block may include incomplete or inaccurate code highlighting.

19.11 Shell Scripts Used (Adam, Jack, James, Shvetank)

19.11.0.1 yagl

```
#!/bin/bash
./yagl.native -c "${1}.ygl" > "${1}.ll"
llc -relocation-model=pic "${1}.ll" > "${1}.s"
cc -c stdlib.c
cc -o "${1}.exe" "${1}.s" stdlib.o
"./${1}.exe"
rm stdlib.o
```

19.11.0.2 testall.sh

Professor Edward's testing suite with minor modifications

```
#!/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 LLVM compiler
LLC="llc"

# Path to the C compiler
CC="cc"

# 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=testall.log
rm -f $globallog
error=0
globalerror=0

keep=0

Usage() {
    echo "Usage: testall.sh [options] [.ygl files]"
    echo "-k    Keep intermediate files"
    echo "-h    Print this help"
    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
    eval $* || {
      SignalError "$1 failed on $*"
      return 1
    }
}

# 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/.ygl//'`
    reffile=`echo $1 | sed 's/.ygl$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."
```

```
    echo -n "$basename..."

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

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.ll ${basename}.s
${basename}.exe ${basename}.out" &&
    Run "$YAGL" "$1" ">" "${basename}.ll" &&
    Run "$LLC" "-relocation-model=pic" "${basename}.ll" ">" "${basename}.s"
&&
    Run "$CC" "-o" "${basename}.exe" "${basename}.s" "stdlib.o" &&
    Run "./${basename}.exe" > "${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
}

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

    echo -n "$basename..."

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

    generatedfiles=""
```

```
    generatedfiles="$generatedfiles ${basename}.err ${basename}.diff
tests/${basename}_preprocessed.ygl" &&
    RunFail "$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
          ;;
    esac
done

shift `expr $OPTIND - 1`

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

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

if [ ! -f stdlib.o ]
then
    # echo "Could not find stdlib.o"
```

```
    # echo "Try \"make stdlib.o\""
    # exit 1
    Run "make" "all"
fi

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

for file in $files
do
    case $file in
      *test-*)
          Check $file 2>> $globallog
          ;;
      *fail-*)
          CheckFail $file 2>> $globallog
          ;;
      *)
          echo "unknown file type $file"
          globalerror=1
          ;;
    esac
done

exit 0
```

19.11.0.3 yagl_debugger

```
#!/bin/bash
make clean && make all
echo "---------------SRC--------------"
cat "${1}.ygl"
echo "---------------AST--------------"
./yagl.native -a "${1}.ygl"
echo "---------------SAST-------------"
./yagl.native -s "${1}.ygl"
echo "---------------LLVM IR----------"
./yagl.native -c "${1}.ygl"
./yagl.native -c "${1}.ygl" > "${1}.ll"
llc -relocation-model=pic "${1}.ll" > "${1}.s"
echo "---------------ASSEMBLY---------"
```

```
cat "${1}.s"
cc -c stdlib.c
cc -o "${1}.exe" "${1}.s" stdlib.o
echo "---------------EXE OUTPUT--------------"
"./${1}.exe"
rm stdlib.o
```

19.12 Testing Code

These test files were automatically run by our testing suite. This was useful to determine if there were any regressions in our code with each commit and merging session.

Each test-x file has a corresponding test-x.out file with expected output. Likewise, each fail-x file has a corresponding .err file with its own expected output. If output for either deviates, the testing suite reports a failure.

test-array2.ygl

```
bool foo() {
  bool[10] bar;
  bar[6] = true;
  return bar[6];
}
bool temp;
temp = foo();
printBool(temp);
```

test-array2.out

```
1
```

test-edge1.ygl

```
Graph G;
Node n1("origin");
Node n2("destination");
G + n1;
G + n2;
G: n1 ->5 n2;
printGraph(G);
```

test-edge1.out

```
============== Graph Print ===============
          Stats:   2         2         1

Nodes:
Node (0) : origin --- Node (1) : destination

Edges:
[(0) : origin] ---(5)---> [(1) : destination]
```

========== End Graph Print ==========

test-graph-node.ygl

```
Graph G;
Node n("Adam");
Node n2("James");
Node n3("Tank");
Node n4("Jack");
G + n;
G + n2;
G + n3;
G + n4;
printGraph(G);
```

test-graph-node.out

```
========== Graph Print ==========
        Stats:  4       4       0

Nodes:
Node (0) : Adam --- Node (1) : James --- Node (2) : Tank
Node (3) : Jack

Edges:
There aren't any edges in this graph.
========== End Graph Print ==========
```

test-binop.ygl

```
printInt(5+10);
printInt(7+15);
```

test-binop.out

```
15
22
```

test-while.ygl

```
int a;
a = 0;
while (a < 3) {
        printInt(a);
        a = a + 1;
}
```

test-while.out

```
0
1
2
```

test-edge6.ygl

```
Graph g_all;
Graph g_add;

Node james("James");
Node jack("Jack");
Node adam("Adam");
Node tank("Tank");
Node random_person("Rando");

g_all: + james + jack -> james + adam ->2 james ->3 jack + tank ->4 adam ->5 jack
      + random_person ->100 tank ->101 random_person - random_person;

printGraph(g_all);

g_add + james;
g_add + jack;
g_add: + tank + adam;
printGraph(g_add);
```

test-edge6.out

```
============== Graph Print ===============
          Stats:   8          4          5

Nodes:
Node (0) : James --- Node (1) : Jack --- Node (2) : Adam
Node (3) : Tank

Edges:
[(1) : Jack] ---(1)---> [(0) : James]
[(2) : Adam] ---(2)---> [(0) : James]
[(2) : Adam] ---(5)---> [(1) : Jack]
[(0) : James] ---(3)---> [(1) : Jack]
[(3) : Tank] ---(4)---> [(2) : Adam]
============ End Graph Print =============
============== Graph Print ===============
          Stats:   4          4          0

Nodes:
Node (0) : James --- Node (1) : Jack --- Node (3) : Tank
Node (2) : Adam

Edges:
There aren't any edges in this graph.
============ End Graph Print =============
```

test-edge3.ygl

```
Graph G;

Node n("Adam");
Node n2("James");
```

```
Node n3("Tank");
Node n4("Jack");
Node n5("Jack2");
Node n6("Jack3");

G + n;
G + n2;
G + n3;
G + n4;
G + n5;
G + n6;

int b = 3;

G: n -> n2 ->2 n3 ->b n4 ->|b + 1| n5 ->|5| n6;

printGraph(G);
```

test-edge3.out

```
============== Graph Print ===============
          Stats:   8         6         5

Nodes:
Node (0) : Adam --- Node (1) : James --- Node (2) : Tank
Node (3) : Jack --- Node (4) : Jack2 --- Node (5) : Jack3


Edges:
[(0) : Adam] ---(1)---> [(1) : James]
[(1) : James] ---(2)---> [(2) : Tank]
[(2) : Tank] ---(3)---> [(3) : Jack]
[(3) : Jack] ---(4)---> [(4) : Jack2]
[(4) : Jack2] ---(5)---> [(5) : Jack3]
============ End Graph Print =============
```

test-stringbinop2.ygl

```
String x = "Hello ";
String y;
y = "World";
printString(x + y + "!");
```

test-stringbinop2.out

```
Hello World!
```

test-edge5.ygl

```
Graph team;

Node adam("Adam");
```

```
Node james("James");
Node tank("Tank");
Node jack("Jack");

team + adam;
team + james;
team + tank;
team + jack;

team: adam -> james -> tank -> jack -> adam;

printGraph(team);
```

test-edge5.out

```
============== Graph Print ===============
           Stats:   4          4          4

Nodes:
Node (0) : Adam --- Node (1) : James --- Node (2) : Tank
Node (3) : Jack

Edges:
[(0) : Adam] ---(1)---> [(1) : James]
[(1) : James] ---(1)---> [(2) : Tank]
[(2) : Tank] ---(1)---> [(3) : Jack]
[(3) : Jack] ---(1)---> [(0) : Adam]
============ End Graph Print =============
```

test-arith2.ygl

```
print(1 + 2 * 3 + 4);
```

test-arith2.out

```
11
```

test-assign.ygl

```
int a;
a=5;
printInt(a);
```

test-assign.out

```
5
```

test-scope3.ygl

```
/* Every single variable is named foo */

String foo = "wins";
{
   String foo = "always";
```

```
  {
    bool foo = true;
    {
      if(foo) {
        int foo = 42;
        printInt(foo);
      }
    }
  }
  printString(foo);

}
printString(foo);
```

test-scope3.out

```
42
always
wins
```

test-edge9.ygl

```
Graph g;

Node n("h");
Node n2("e");
Node n3("l");
Node n4("o");
/*
g: + n + n2 <-> n & + n3 <-> n2 & n3 <-> n & + n4 <-> n3
  & n4 <-> n2 & n4 <-> n;

equivalent to: */

g: + n + n2 + n3 + n4;
g: n <-> n2;
g: n <-> n3;
g: n <-> n4;
g: n2 <-> n4;
g: n2 <-> n3;
g: n3 <-> n4;

printGraph(g);
```

test-node2.ygl

```
Node n;
n = "hi";
printNode(n);
```

test-node2.out

```
hi
```

test-edge9.out

```
============== Graph Print ===============
            Stats:   4          4          4

Nodes:
Node (0) : h --- Node (1) : e --- Node (2) : l
Node (3) : o

Edges:
[(0) : h] ---(1)---> [(1) : e]
[(0) : h] ---(1)---> [(2) : l]
[(0) : h] ---(1)---> [(3) : o]
[(1) : e] ---(1)---> [(0) : h]
[(1) : e] ---(1)---> [(3) : o]
[(1) : e] ---(1)---> [(2) : l]
[(2) : l] ---(1)---> [(0) : h]
[(2) : l] ---(1)---> [(1) : e]
[(2) : l] ---(1)---> [(3) : o]
[(3) : o] ---(1)---> [(0) : h]
[(3) : o] ---(1)---> [(1) : e]
[(3) : o] ---(1)---> [(2) : l]
============ End Graph Print =============
```

test-while2.ygl

```
int foo(int a)
{
  int j;
  j = 0;
  while (a > 0) {
    j = j + 2;
    a = a - 1;
  }
  return j;
}

  print(foo(7));
  return 0;
```

test-while2.out

```
14
```

test-get_weight.ygl

```
Graph g;
Node n("hi");
Node n2("hi2");

g: + n + n2 [5<->8] n;
g: n2 [10<->12] n;
```

```
int weight = g.weight[n,n2];
print(weight);
```

test-get_weight.out

```
10
```

test-array_nodes.ygl

```
Graph g;
Node n("hi");
g: +n;
Node[5] x;
x[0] = n;
print(x[0]);
```

test-array_nodes.out

```
hi
```

test-ops1.ygl

```
  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);
  return 0;
```

test-ops1.out

```
3
-1
2
50
99
```

0
1
99
1
0
99
1
0
99
1
1
0
99
0
1
99
0
1
1

test-copy_graph.ygl

```
Graph g;
Node n1("Hi there");
Node n2("hey");
Node n3("Bye");
g: + n1 + n2 + n3 -> n2 <-5 n1;
Graph g2;
g2 = copy_graph_lib(g);
print("original:");
print(g);
print("Copied:");
print(g2);
Graph copy_graph_lib(Graph g) {

	Graph g2;

	int nodes = g.num_nodes;
	int curr = 0;
	while (curr < nodes) {
		Node current;
		current = g.node[curr];
		g2: + current;
		curr = curr + 1;
	}

	curr = 0;
	while (curr < nodes) {

		Node current = g.node[curr];
```

```
                int neighs = g.num_neighbors[current];
                int on = 0;
                while (on < neighs) {
                        Node to;
                        to  = g.neighbor[current, on];
                        int val = g.weight[current, to];
                        g2: current ->val to;
                        on = on + 1;
                }

                curr = curr + 1;
        }

        return g2;

}
```

test-copy_graph.out

```
original:
============== Graph Print ===============
        Stats:   4       3       2

Nodes:
Node (0) : Hi there --- Node (1) : hey --- Node (2) : Bye


Edges:
[(2) : Bye] ---(1)---> [(1) : hey]
[(0) : Hi there] ---(5)---> [(1) : hey]
============ End Graph Print =============
Copied:
============== Graph Print ===============
        Stats:   4       3       2

Nodes:
Node (0) : Hi there --- Node (1) : hey --- Node (2) : Bye


Edges:
[(0) : Hi there] ---(5)---> [(1) : hey]
[(2) : Bye] ---(1)---> [(1) : hey]
============ End Graph Print =============
```

test-edge2.ygl

```
Graph G;
Node n1("origin");
Node n2("destination");
G + n1;
G + n2;
```

```
G: n1 -> n2;
printGraph(G);
```

test-edge2.out

```
============== Graph Print ===============
            Stats:   2          2          1

Nodes:
Node (0) : origin --- Node (1) : destination

Edges:
[(0) : origin] ---(1)---> [(1) : destination]
============ End Graph Print =============
```

test-node3.ygl

```
String init = "This is x";
Node x(init);
printNode(x);
String updt = "not!";
x = updt;
printNode(x);
```

test-node3.out

```
This is x
not!
```

test-while1.ygl

```
  int i;
  i = 5;
  while (i > 0) {
    print(i);
    i = i - 1;
  }
  print(42);
  return 0;
```

test-while1.out

```
5
4
3
2
1
42
```

test-library-copy.ygl

```
void reset(Graph g) {
    int size = g.num_nodes;
    int curr = 0;
```

```
    while (curr < size) {
      Node current;
      current = g.node[curr];
      current.visited = false;
      current.curr_dist = 0;
      curr = curr + 1;
    }
}

Graph g;
Node n1("Hi there");
Node n2("hey");
Node n3("Bye");
g: + n1 + n2 + n3 -> n2 <-5 n1;
Graph g2;
g2 = copy_graph_lib(g);
print("original:");
print(g);
print("Copied:");
print(g2);
Graph copy_graph_lib(Graph g) {

        Graph g2;

        int nodes = g.num_nodes;
        int curr = 0;
        while (curr < nodes) {
                Node current = g.node[curr];
                g2: + current;
                curr = curr + 1;
        }

        curr = 0;
        while (curr < nodes) {

                Node current = g.node[curr];
                int neighs = g.num_neighbors[current];
                int on = 0;
                while (on < neighs) {
                        Node to;
                        to  = g.neighbor[current, on];
                        int val = g.weight[current, to];
                        g2: current ->val to;
                        on = on + 1;
                }

                curr = curr + 1;
        }

        return g2;
```

}

test-library-copy.out
original:
============== Graph Print ==============
Stats: 4 3 2

Nodes:
Node (0) : Hi there --- Node (1) : hey --- Node (2) : Bye

Edges:
[(2) : Bye] ---(1)---> [(1) : hey]
[(0) : Hi there] ---(5)---> [(1) : hey]
============ End Graph Print ============
Copied:
============== Graph Print ==============
Stats: 4 3 2

Nodes:
Node (0) : Hi there --- Node (1) : hey --- Node (2) : Bye

Edges:
[(0) : Hi there] ---(5)---> [(1) : hey]
[(2) : Bye] ---(1)---> [(1) : hey]
============ End Graph Print ============

test-array3.ygl
```
String[3] strs;
strs[0] = "Hi,";
strs[1] = "my name is";
strs[2] = "Bob";
int temp;
temp = 2;
printString(strs[0]);
printString(strs[1]);
printString(strs[temp]);
```

test-array3.out
Hi,
my name is
Bob

test-declareassign.ygl
```
int test(int q) {
int x = 5;
int y;
```

```
y = 10;
int z = x * 2 + y;
printInt(z + q);
}

test(1);
test(5);
test(10);
```

test-declareassign.out

```
21
25
30
```

test-if3.ygl

```
if (false)
	printBool(false);
printBool(true);
```

test-if3.out

```
1
```

test-visited2.ygl

```
Node n("unvisited");
n.visited = true;
if (n.visited) {
	printBool(true);
}
```

test-visited2.out

```
1
```

t**est-visited3.ygl**

```
Node n1("n1");
Node n2("n2");
Node n3("n3");

n1.visited = true;
n3.visited = true;

printBool(n1.visited);
printBool(n2.visited);
printBool(n3.visited);
```

test-visited3.out

```
1
0
```

1

test-fib.ygl

```
int fib(int x)
{
  if (x < 2) return 1;
  return fib(x-1) + fib(x-2);
}

  print(fib(0));
  print(fib(1));
  print(fib(2));
  print(fib(3));
  print(fib(4));
  print(fib(5));
```

test-fib.out

```
1
1
2
3
5
8
```

test-curr_dist.ygl

```
Node n1("n1");
printInt(n1.curr_dist);
while (n1.curr_dist < 3) {
        n1.curr_dist = n1.curr_dist + 1;
        printInt(n1.curr_dist);
}
```

test-curr_dist.out

```
0
1
2
3
```

test-visited.ygl

```
Node n("unvisited");
if (n.visited) {
        printBool(true);
}
printBool(false);
```

test-visited.out

```
0
```

test-if1.ygl

```
if (true)
	printBool(true);
```

test-if1.out
1

test-graph_size.ygl
```
Graph g;
Node n("hi");
Node n2("hi");
Node n3("hi");
g: + n + n2 + n3;
int x = g.num_nodes;
printInt(x);
```

test-graph_size.out
3

test-get_node.ygl
```
Node n("test");
Node n2("test2");
Graph g;
g: + n + n2;

int num = g.num_nodes;
int pos = 0;
while (pos < num) {
	print(g.node[pos]);
	pos = pos + 1;
}
```

test-get_node.out
test
test2

test-hello.ygl
```
printString("Hello World!");
printInt(0);
```

test-hello.out
Hello World!
0

test-demo2.ygl
```
started();
void finished() {
	printString("Executed" + " Completely");
```

```
        printString("******************");
}
String[3] foo;
String[3] t;
t[0] = "Make";
t[1] = "Some";
t[2] = "Noise";
int bar = 0;
String x = "";
while(bar < 3) {
        foo[bar] = t[bar];
        x = x + " " + foo[bar];
        printString(x);
        bar = bar + 1;
}
finished();
void started() {
        printString("******************");
        printString("Loading" + " ...");
}
```

test-demo2.out

```
******************
Loading ...
 Make
 Make Some
 Make Some Noise
Executed Completely
******************
```

test-char.ygl

```
char a = 'c' ;
char b = '\n' ;
printChar(a);
printChar(b);
```

test-char.out

```
c
```

test-import.ygl

```
/* Preprocessing directives */
import stdgraph.ygl
import stdalgo.ygl

/* Social Media Application */

Graph fb;       /* Facebook  */

/* People */
Node adam("Adam");
```

```
Node james("Jamie");                                    /* Typo on purpose for demonstration later
*/
Node jack("Jack");
Node tank("Shvetank");


/* Add people to Facebook's network */
fb: + adam + james + jack + tank;

/* Add edges between people to make them friends */
fb: adam <-> james <-> jack, tank <-> jack, tank <-> adam;          /* Facebook friends is a
bidirectional relationship */

/* Find friends of friends */
dfs(fb, james, 2);          /* Should find Shvetank, Adam, and James */
print(fb);
```

test-import.out

```
Adam
Shvetank
Jack
============== Graph Print ===============
          Stats:   4          4          4

Nodes:
Node (0) : Adam --- Node (1) : Jamie --- Node (2) : Jack
Node (3) : Shvetank

Edges:
[(0) : Adam] ---(1)---> [(1) : Jamie]
[(0) : Adam] ---(1)---> [(3) : Shvetank]
[(1) : Jamie] ---(1)---> [(0) : Adam]
[(1) : Jamie] ---(1)---> [(2) : Jack]
[(2) : Jack] ---(1)---> [(1) : Jamie]
[(2) : Jack] ---(1)---> [(3) : Shvetank]
[(3) : Shvetank] ---(1)---> [(2) : Jack]
[(3) : Shvetank] ---(1)---> [(0) : Adam]
============ End Graph Print =============
```

test-stringbinop.ygl

```
printString("Hello" + " World");
```

test-stringbinop.out

```
Hello World
```

test-edge4.ygl

```
Graph G;
```

```
Node n("Adam");
Node n2("James");
Node n3("Tank");
Node n4("Jack");

G + n;
G + n2;
G + n3;
G + n4;

int b = 3;

/* All different types of edges */

G: n -> n2;                 /* Default 1 */
G: n ->2 n3;                /* Simply just a literal */
G: n2 ->b n3;               /* Simply just a variable */
G: n3 ->|b + 1| n2;         /* Expression */
G: n3 ->|5| n4;             /* Optional | */

printGraph(G);
```

test-edge4.out

```
============== Graph Print ===============
          Stats:   4         4         3

Nodes:
Node (0) : Adam --- Node (1) : James --- Node (2) : Tank
Node (3) : Jack

Edges:
[(0) : Adam] ---(1)---> [(1) : James]
[(0) : Adam] ---(2)---> [(2) : Tank]
[(1) : James] ---(3)---> [(2) : Tank]
[(2) : Tank] ---(4)---> [(1) : James]
[(2) : Tank] ---(5)---> [(3) : Jack]
============ End Graph Print =============
```

test-edge10.ygl

```
Graph demo;

Node portland("Portland");
Node pittsburgh("Pittsburgh");
Node nyc("New York City");
Node nj("New Jersey");

demo: + portland + pittsburgh ->2566 portland ->2566 pittsburgh
      + nyc [800<->800] pittsburgh, portland [2900<->2900] nyc,
      + nj <-13 nyc <-13 nj [2899<->2899] portland,
```

```
    nj [790<->790] pittsburgh;

print(demo);
```

test-edge10.out

```
============== Graph Print ==============
          Stats:   4         4         4

Nodes:
Node (0) : Portland --- Node (1) : Pittsburgh --- Node (2) : New York City
Node (3) : New Jersey

Edges:
[(1) : Pittsburgh] ---(2566)---> [(0) : Portland]
[(1) : Pittsburgh] ---(800)---> [(2) : New York City]
[(1) : Pittsburgh] ---(790)---> [(3) : New Jersey]
[(0) : Portland] ---(2566)---> [(1) : Pittsburgh]
[(0) : Portland] ---(2900)---> [(2) : New York City]
[(0) : Portland] ---(2899)---> [(3) : New Jersey]
[(2) : New York City] ---(800)---> [(1) : Pittsburgh]
[(2) : New York City] ---(2900)---> [(0) : Portland]
[(2) : New York City] ---(13)---> [(3) : New Jersey]
[(3) : New Jersey] ---(13)---> [(2) : New York City]
[(3) : New Jersey] ---(2899)---> [(0) : Portland]
[(3) : New Jersey] ---(790)---> [(1) : Pittsburgh]
============ End Graph Print =============
```

test-library-reverse.ygl

```
void reset(Graph g) {
   int size = g.num_nodes;
   int curr = 0;

   while (curr < size) {
     Node current;
     current = g.node[curr];
     current.visited = false;
     current.curr_dist = 0;
     curr = curr + 1;
   }
}

Graph g;
Node n1("Hi there");
Node n2("hey");
Node n3("Bye");
g: + n1 + n2 + n3 -> n2 <-5 n1;
print("original:");
print(g);
g = reverse_graph_lib(g);
print("reverse:");
```

```
print(g);
Graph reverse_graph_lib(Graph g) {

	Graph g2;

	int nodes = g.num_nodes;
	int curr = 0;
	while (curr < nodes) {
		Node current = g.node[curr];
		g2: + current;
		curr = curr + 1;
	}

	curr = 0;
	while (curr < nodes) {

		Node current = g.node[curr];
		int neighs = g.num_neighbors[current];
		int on = 0;
		while (on < neighs) {
			Node to;
			to  = g.neighbor[current, on];
			int val = g.weight[current, to];
			g2: to ->val current; /* Reversed */
			on = on + 1;
		}

		curr = curr + 1;
	}
	return g2;
}
```

test-library-reverse.out

```
original:
============== Graph Print ===============
	Stats:   4        3        2

Nodes:
Node (0) : Hi there --- Node (1) : hey --- Node (2) : Bye


Edges:
[(2) : Bye] ---(1)---> [(1) : hey]
[(0) : Hi there] ---(5)---> [(1) : hey]
============ End Graph Print =============
reverse:
============== Graph Print ===============
	Stats:   4        3        1

Nodes:
```

Node (0) : Hi there --- Node (1) : hey --- Node (2) : Bye

Edges:
[(1) : hey] ---(5)---> [(0) : Hi there]
[(1) : hey] ---(1)---> [(2) : Bye]
========== End Graph Print ==========

test-node.ygl
```
Node a ("hello");
printNode(a);
```

test-node.out
```
hello
```

test-curr_dist2.ygl
```
Node n("n");
n.curr_dist = 5;
n.curr_dist = n.curr_dist + 5;
printInt(n.curr_dist);
```

test-curr_dist2.out
```
10
```

test-edge7.ygl
```
Graph g;

Node n("h");
Node n2("e");
Node n3("l");

int b = 6;

/* Must use [ ] for bidirectional with
   separate weights in recursion*/
g: + n + n2 + n3 [7<->5] n2;

/* [ ] optional for bidir with separate
   weights with no recursion*/
g: n2 6<->9 n;
g: n |b+10|<->b n3;

printGraph(g);
```

test-edge7.out
```
========== Graph Print ==========
```

```
          Stats:   4          3          3

Nodes:
Node (0) : h --- Node (1) : e --- Node (2) : l


Edges:
[(2) : l] ---(5)---> [(1) : e]
[(2) : l] ---(16)---> [(0) : h]
[(1) : e] ---(7)---> [(2) : l]
[(1) : e] ---(9)---> [(0) : h]
[(0) : h] ---(6)---> [(1) : e]
[(0) : h] ---(6)---> [(2) : l]
============ End Graph Print =============
```

test-float.out

```
7.1
Hello World!
6.2
./fail-func2.err
Fatal error: exception Failure("duplicate function foo")
```

test-float.out

```
7.1
Hello World!
6.2
```

test-gcd.ygl

```
int gcd(int a, int b) {
  while (a < b || b < a) {
    if (a > b) {  a = a - b; }
    else { b = b - a; }
  }
  return a;
}

  print(gcd(2,14));
  print(gcd(3,15));
  print(gcd(99,121));
```

test-gcd.out

```
2
3
11
```

test-string-length.ygl

```
float a;

int printHello() {
```

```
  String hello;

  hello = "Hello World";

  printString(hello);
  printInt(hello.length);
}


printHello();
```

test-string-length.out

```
Hello World
11
```

test-edge8.ygl

```
Graph g;

Node n("h");
Node n2("e");
Node n3("l");
Node n4("o");

g: + n + n2 <-> n, + n3 <-> n2, n3 <-> n, + n4 <-> n3,
    n4 <-> n2, n4 <-> n;

/* equivalent to:

g: + n + n2 + n3 + n4;
g: n <-> n2;
g: n <-> n3;
g: n <-> n4;
g: n2 <-> n4;
g: n2 <-> n3;
g: n3 <-> n4; */

printGraph(g);
```

test-edge8.out

```
============== Graph Print ===============
          Stats:   4          4          4

Nodes:
Node (0) : h --- Node (1) : e --- Node (2) : l
Node (3) : o

Edges:
[(1) : e] ---(1)---> [(0) : h]
```

[(1) : e] ---(1)---> [(2) : l]
[(1) : e] ---(1)---> [(3) : o]
[(0) : h] ---(1)---> [(1) : e]
[(0) : h] ---(1)---> [(2) : l]
[(0) : h] ---(1)---> [(3) : o]
[(2) : l] ---(1)---> [(1) : e]
[(2) : l] ---(1)---> [(0) : h]
[(2) : l] ---(1)---> [(3) : o]
[(3) : o] ---(1)---> [(2) : l]
[(3) : o] ---(1)---> [(1) : e]
[(3) : o] ---(1)---> [(0) : h]
========== End Graph Print ==========

test-generic_print.ygl

```
Node n("hi");
print(n);
int x = 5;
print(x);
char s = 's';
print(s);
print('s');
Graph g;
g: + n;
print(g);
print("Hi!!!");
print(true);
```

test-generic_print.out

```
hi
5
s
s
============== Graph Print ===============
            Stats:   1          1          0

Nodes:
Node (0) : hi

Edges:
There aren't any edges in this graph.
========== End Graph Print ==========
Hi!!!
1
```

test-var1.ygl

```
  int a;
  a = 42;
  print(a);
```

test-var1.out

```
42
```

test-scope2.ygl

```
int a;
a= 5;
int c;
c= 3;
if (true) {

  int b = 7;
  String c = "abc";

  {
    int d;
    d = 25;
    printInt(d);
  }

  printInt(b);
  printInt(a);
  printString(c);
}

printInt(a);
printInt(c);
```

test-scope2.out

```
25
7
5
abc
5
3
```

test-scope1.ygl

```
int a;
a= 5;
int c;
c= 3;
if (true) {

  int b = 7;
  String c = "abc";

  if(true){
    int d;
    d = 25;
```

```
    printInt(d);
  }

  printInt(b);
  printInt(a);
  printString(c);
}
printInt(a);
printInt(c);
```

test-scope1.out

```
25
7
5
abc
5
3
```

test-node4.ygl

```
Node y("test");
Node x;
x = y;
printNode(x);
x = "hi";
printNode(y);
```

test-node4.out

```
test
hi
```

test-bools.ygl

```
bool b;

bool foo(bool bar) {
  printBool(bar);
}

printBool(false);
foo(true);
```

test-bools.out

```
0
1
```

test-add1.ygl

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

  print( add(17, 25) );
  return 0;
```

test-add1.out

```
42
```

test-get_neighbors.ygl

```
Node n("James");
Node n2("Tank");
Node n3("Adam");
Node n4("Jack");

Graph g;
g: +n + n2 <-> n + n3 + n4 <-> n3 <-> n;

print_neighbors(g, n);

void print_neighbors(Graph g, Node n) {

        int num = g.num_neighbors[n];
        int pos = 0;
        while (pos < num) {
                print(g.neighbor[n,pos]);
                pos = pos + 1;
        }

}
```

test-get_neighbors.out

```
Tank
Adam
```

test-get_node_name.ygl

```
Node n("James");
printString(n.name);
```

test-get_node_name.out

```
James
```

test-if4.ygl

```
if (false)
        printBool(false);
```

```
else
	printBool(true);
```

test-if4.out

```
1
```

test-decls.ygl

```
int a;

int printHello(int a) {
  printString("Hello World!");
  printInt(a);
}

int printHello2() {
  int a;

  printInt(7);
  printHello(5);
}


printHello2();
```

test-decls.out

```
7
Hello World!
5
```

test-demo1.ygl

```
started();
void finished() {
	printString("Executed" + " Completely");
	printString("********************");
}
String[3] foo;
String[3] t;
t[0] = "Make";
t[1] = "Some";
t[2] = "Noise";
int bar = 0;
while(bar < 3) {
	foo[bar] = t[bar];
	printString(foo[bar]);
	bar = bar + 1;
}
finished();
void started() {
	printString("******************");
```

```
	printString("Loading" + " ...");
}
```

test-demo1.out

```
******************
Loading ...
Make
Some
Noise
Executed Completely
******************
```

test-array.ygl

```
int bar;
bar = 9;

int[10] foo;
foo[0] = 0;
foo[2+3] = 123;
foo[bar] = 456;

printInt(foo[3-3]);
printInt(foo[5]);
printInt(foo[bar]);

int temp;
temp = foo[0];
printInt(temp);
```

test-array.out

```
0
123
456
0
```

test-return.ygl

```
printString(sreturned(55));
String sreturned(int x) {
	printInt(x);
	return "Hello World Returned!";
}

int ireturned(int x) {
	return x;
}

printInt(ireturned(5559));
```

test-return.out

```
55
```

Hello World Returned!
5559

test-if2.ygl

```
if (true)
	printBool(true);
else
	printBool(false);
```

test-if2.out

```
1
```

test-arith1.ygl

```
 print(39 + 3);
```

test-arith1.out

```
42
```

test-demo3.ygl

```
/* Every single variable is named foo */

Graph foo;
{
  Node bar("Michael1");
  foo + bar;
  Node bar2("Michael2");
  foo + bar2;
  foo: bar ->1 bar2;
  int i = 0;
  while (i < 3) {
	if (i == 2) {
		Graph foo;
		Node bar("Finished");
		foo + bar;
		foo: bar2 -> bar;
		foo + bar2;
		foo: bar2 ->5 bar;
		printGraph(foo);
	} else {
		if (i == 1) {
			Graph foo;
			Node bar("Michael5");
			foo + bar;
			Node bar2("Michael6");
			foo + bar2;
			foo: bar -> bar2;
			printGraph(foo);
		} else {
```

```
                                Graph foo;
                                Node bar("Michael3");
                                foo + bar;
                                Node bar2("Michael4");
                                foo + bar2;
                                foo: bar -> bar2;
                                printGraph(foo);
                        }
                }
                i = i + 1;
    }
}
printGraph(foo);
```

test-demo3.out

```
============== Graph Print ==============
        Stats:  2       2       1

Nodes:
Node (2) : Michael3 --- Node (3) : Michael4

Edges:
[(2) : Michael3] ---(1)---> [(3) : Michael4]
============ End Graph Print =============
============== Graph Print ==============
        Stats:  2       2       1

Nodes:
Node (4) : Michael5 --- Node (5) : Michael6

Edges:
[(4) : Michael5] ---(1)---> [(5) : Michael6]
============ End Graph Print =============
Nodes don't exist in given graph.
============== Graph Print ==============
        Stats:  2       2       1

Nodes:
Node (6) : Finished --- Node (1) : Michael2

Edges:
[(1) : Michael2] ---(5)---> [(6) : Finished]
============ End Graph Print =============
============== Graph Print ==============
        Stats:  2       2       1

Nodes:
Node (0) : Michael1 --- Node (1) : Michael2

Edges:
```

[(0) : Michael1] ---(1)---> [(1) : Michael2]
========== End Graph Print ==========

./test-decls.out
7
Hello World!
5

./test-edge4.ygl
```
Graph G;

Node n("Adam");
Node n2("James");
Node n3("Tank");
Node n4("Jack");

G + n;
G + n2;
G + n3;
G + n4;

int b = 3;

/* All different types of edges */

G: n -> n2;                 /* Default 1 */
G: n ->2 n3;                /* Simply just a literal */
G: n2 ->b n3;               /* Simply just a variable */
G: n3 ->|b + 1| n2;         /* Expression */
G: n3 ->|5| n4;             /* Optional | */

printGraph(G);
```

fail-edge2.ygl

```
Graph G;
Node n1("origin");
Node n2("destination");
G + n1;
G + n2;
n1: G -> n2; /* should be G: n1 -> n2 */
printGraph(G);
```

fail-edge2.err

```
Fatal error: exception Failure("illegal graph operator Graph ->{int} Node in G ->|1| n2")
```

fail-visited2.ygl

```
Graph g;
g.visited; /*should be node not graph */
```

fail-visited2.err

```
Fatal error: exception Failure("Wrong accessor type, [void, void], on attribute visited.")
```

fail-scope1.ygl

```
int a;
a= 5;
int c;
c= 3;
if (true) {

  int b = 7;
  String c = "abc";

  if(true){
    int d;
    d = 25;
  }

  printInt(d);
}
```

fail-scope1.err

```
Fatal error: exception Failure("undeclared identifier d")
```

fail-string-length.ygl

```
int a = 1;

printInt(a.length);
```

fail-string-length.err

```
Fatal error: exception Failure("Wrong accessor type, [void, void], on attribute length.")
```

fail-return.ygl

```
String sreturned(int x) {
        printInt(x);
        return 0;
}
```

fail-return.err

```
Fatal error: exception Failure("return gives int, but expected String in return 0")
```

fail-array6.ygl

```
bool[5] a;
a[-1] = true; /* Index out of bounds */
```

fail-array6.err

```
Fatal error: exception Failure("ERROR: Index out of bounds.")
```

fail-defineownmain.ygl

```
int main2() {
printString("Hello World!");
```

```
printInt(0);
}
main2();

String main() {
}
```

fail-defineownmain.err
Fatal error: exception Failure("reserved function name: main cannot be used")

fail-visited.ygl
```
Graph g;
g.visited = true; /* should be node not graph */
```

fail-visited.err
Fatal error: exception Failure("illegal visit operands Graph bool in g visited true")

fail-bools.ygl
```
bool b;

bool foo() {
  printBool(0);
}

foo();
```

fail-bools.err
Fatal error: exception Failure("illegal argument found int expected bool in 0")

fail-get_neighbors.ygl
```
Node n("James");
Node n2("Tank");
Node n3("Adam");
Node n4("Jack");

Graph g;
g: +n + n2 <-> n + n3 + n4 <-> n3 <-> n;

print_neighbors(g, n);

void print_neighbors(Graph g, Node n) {

        int num = g.num_neighbors[n];
        int pos = 0;
        while (pos < num) {
                print(g.neighbor[n,n]);
                pos = pos + 1;
        }
```

```
}
```

fail-get_neighbors.err
Fatal error: exception Failure("Wrong accessor type, [Node, Node], on attribute neighbor.")

fail-array.ygl
```
int[true] a; /* Invalid expr to have in braces */
```

fail-array.err
Fatal error: exception Failure("Size of array is not of type int.")

fail-assign.ygl
```
int a;
a = "abc";
printInt(a);
```

fail-assign.err
Fatal error: exception Failure("illegal assignment int = String in a = abc")

fail-binop.ygl
```
printInt(5+10);
printInt(7+"2");
```

fail-binop.err
Fatal error: exception Failure("illegal binary operator int + String in 7 + 2")

fail-node.ygl
```
Node x("");
int y = 5;
x = y;
```

fail-node.err
Fatal error: exception Failure("illegal assignment Node = int in x = y")

fail-func2.err
Fatal error: exception Failure("duplicate function foo")

fail-func2.ygl
```
int foo() {

}

int foo() {

}
```

fail-hello.ygl
```
printString(5);
```

fail-hello.err

Fatal error: exception Failure("illegal argument found int expected String in 5")

fail-func1.ygl
```
int foo() {}

int bar() {}

int baz() {}

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

int main()
{
  return 0;
}
```

fail-func1.err
Fatal error: exception Failure("reserved function name: main cannot be used")

fail-stringbinop.ygl
```
printString("Hi" + 5);
```

fail-stringbinop.err
Fatal error: exception Failure("illegal binary operator String + int in Hi + 5")

fail-decls.ygl
```
int a;

int printHello{int a) {
  printString("Hello World!");
  printInt(a);
}

int printHello2() {
  int a;

  printInt(7);
  printHello(5);
}

printHello2();
```

fail-decls.err
Fatal error: exception Parsing.Parse_error

fail-import.ygl
```
/* FORGET Preprocessing directives */

/* Social Media Application */
```

```
Graph fb;       /* Facebook  */

/* People */
Node adam("Adam");
Node james("Jamie");                                    /* Typo on purpose for demonstration later
*/
Node jack("Jack");
Node tank("Shvetank");


/* Add people to Facebook's network */
fb: + adam + james + jack + tank;

/* Add edges between people to make them friends */
fb: adam <-> james <-> jack, tank <-> jack, tank <-> adam;              /* Facebook friends is a
bidirectional relationship */

/* Find friends of friends */
dfs(fb, james, 2);          /* Should FAIL to find dfs bc import of library forgotten */
print(fb);
```

fail-import.err

```
Fatal error: exception Sys_error("james,: No such file or directory")
```

fail-declareassign.ygl

```
int test(int q) {
int x = 5;
int y;
y = 10;
int z = x * 2 + y;
printInt(z + q);
}

test(1);
test(5);
test(10);

int qqq = "hi";
```

fail-declareassign.err

```
Fatal error: exception Failure("illegal assignment int = String in qqq = hi")
```

fail-edge1.ygl

```
Graph G;
Node n1("origin");
Node n2("destination");
G + n1;
G + n2;
```

```
G: n1 ->|"five"| n2; /* "five" should be an int */
printGraph(G);
```

fail-edge1.err

```
Fatal error: exception Failure("illegal graph operator Node ->{String} Node in n1 ->|five| n2")
```

fail-assign.ygl

```
int a;
a = "abc";
printInt(a);
```

fail-assign.err

```
Fatal error: exception Failure("illegal assignment int = String in a = abc")
```

fail-binop.ygl

```
printInt(5+10);
printInt(7+"2");
```

fail-binop.err

```
Fatal error: exception Failure("illegal binary operator int + String in 7 + 2")
```

fail-while.ygl

```
int a;
a = 2;
while (a) { /* should be boolean not int */
        printInt(a);
}
```

fail-while.err

```
Fatal error: exception Failure("expected Boolean expression in a")
```

fail-node.ygl

```
Node x("");
int y = 5;
x = y;
```

fail-node.err

```
Fatal error: exception Failure("illegal assignment Node = int in x = y")
```

fail-array5.ygl

```
bool[5] a;
a[5] = false; /* Index out of bounds */
```

fail-array5.err

```
Fatal error: exception Failure("ERROR: Index out of bounds.")
```

fail-callmain.ygl

```
int main2() {
}
main2();
main();
```

fail-callmain.err
Fatal error: exception Failure("Cannot call main otherwise recurse forever")

fail-curr_dist.ygl

```
Node n("n");
bool x = n.curr_dist; /* x should be an int */
printBool(x);
```

fail-curr_dist2.err
Fatal error: exception Failure("illegal binary operator int + float in n.curr_dist + 5.5")

fail-dupdefine.ygl

```
int main2() {
printString("Hello World!");
printInt(0);
}
main2();

int main2() {
printString("Hello World!");
printInt(0);
}
```

fail-dupdefine.err
Fatal error: exception Failure("duplicate function main2")

fail-scope2.ygl

```
int a;
a= 5;
int c;
c= 3;
{

  int b = 7;
  String c = "abc";

  {
    int d;
    d = 25;
```

```
  }

  printInt(d);
}
```

fail-scope2.err
Fatal error: exception Failure("undeclared identifier d")

fail-visited3.ygl
```
Node n("n");
n.visited = 5; /* should be bool not int */
```

fail-visited3.err
Fatal error: exception Failure("illegal visit operands Node int in n visited 5")

fail-edge3.ygl
```
Graph team;

Node adam("Adam");
Node james("James");
Node tank("Tank");
Node jack("Jack");

team + adam;
team + james;
team + tank;
team + jack;

int x;
x= 5;
x: adam -> james -> tank -> jack -> adam;

printGraph(team);
```

fail-edge3.err
Fatal error: exception Failure("illegal graph operator: int x. Was expecting type Graph")

fail-float.ygl
```
float a;

int printHello(float a) {
  printString("Hello World!");
  printFloat(a);
}
```

```
int printHello2() {
  float a;

  printFloat(7);
  printHello(6.2);
}


printHello2();
```

fail-float.err
Fatal error: exception Failure("illegal argument found int expected float in 7")

fail-array2.ygl
```
bool[5] a;
a[true]; /* Can not index with non Int type */
```

fail-array2.err
Fatal error: exception Failure("Can only access array element with int type.")

fail-if.ygl
```
if (2)
        printInt(2); /* should be boolean not int */
```
fail-if.err
Fatal error: exception Failure("expected Boolean expression in 2")

fail-edge4.ygl
```
Graph team;

Node adam("Adam");
Node james("James");
Node tank("Tank");
Node jack("Jack");

team + adam;
team + james;
team + tank;
team + jack;

String s_james = "james";
team: adam -> s_james -> tank -> jack -> adam;

printGraph(team);
```

fail-edge4.err
Fatal error: exception Failure("illegal graph operator String ->{int} Node in s_james ->|1| tank")

fail-char.ygl

```
char foo = '\z';
```

fail-char.err
Fatal error: exception Failure("illegal character \'")

fail-array3.ygl
```
bool[5] a;
a[0] = "String"; /* Invalid type assigned */
```

fail-array3.err
Fatal error: exception Failure("illegal assignment bool = String in a[0] = String")

fail-scope3.ygl
```
/* Every single variable is named foo */

String foo = "wins";
{
  String foo = "always";
  {
    bool foo = true;
    {
      if(foo) {
        int foo = 42;
        printInt(foo);
      }
    }
  }
  printInt(foo); /* Try to print inner foo which is an int but out of scope */

}
printString(foo);
```

fail-scope3.err
Fatal error: exception Failure("illegal argument found String expected int in foo")