yeezyGraph Project Report

An Easy Graph Language

Nancy Xu (nx2131), Wanlin Xie (wx2161), Yiming Sun (ys2832)

1 Introduction

The yeezyGraph language is a concise, domain-specific language that is specifically designed for graph data analysis. In practice, graphs can be represented using all kinds of data structures – among them adjacency lists, adjacency matrices, as well as incidence matrices – with different graph implementations varying widely in efficiency and performance. yeezyGraph's unique language constructs make it intuitive for users to leverage its syntactic and structural simplicity to create and manipulate graph objects without having to consider the underlying low-level implementation details. Under the hood, there is a standardized management of graph data that is specifically tailored for efficient graph data analysis.

yeezyGraph's out-of-the-box framework is thus expected to provide users with a powerful and versatile tool to solve a wide variety of graph-based problems, including search, routing, and other fundamental graph algorithms. yeezyGraph is compiled to the LLVM (Low Level Virtual Machine) intermediate form, which can then be optimized to machine-specific assembly code.

2 Language Tutorial

2.1 Compiler Installation and Setup

1. Unpack the yeezyGraph.tar.gz file

2. Inside the yeezyGraph folder, to create the yeezyGraph to LLVM compiler yeezyGraph.native, type:
   make

3. To just generate the internal LLVM modules for yeezyGraph's internal data structures, type:
   ./generateLLVMModules.sh

4. To compile and run a program written in the yeezyGraph language, type:
   ./yeezyGraph.sh <examplecode/filename.yg> a.ll
   llvm-link linkedlist.bc queue.bc pqueue.bc map.bc node.bc graph.bc a.ll –S > run.ll
   clang run.ll
   ./a.out

2.2 Code Example

This section demonstrates several key language features of yeezyGraph, using a simple code example.

2.2.1 Creating a Source File: example.yg

The main() function is the entry point of any yeezyGraph program, where the execution of the program is started. Every yeezyGraph program must have a single, mandatory main() function.

```
int main()
{
  return 0;
}
```

2.2.2 Declaring and Assigning Variables

Variables must be declared before they can be assigned. In yeezyGraph, variables are declared by stating the variable type, followed by the variable name.

```
string a1;
string a2;

a1 = "a1";
a2 = "a2";
```

2.2.3 Nodes and Graphs

Nodes and graphs are the fundamental data types in yeezyGraph. A graph is a collection of nodes. A node, meanwhile, is a data type that contains five different fields: name, visited, inNodes, outNodes, and data (see Section 3.2.3.2).

To declare a graph of type parameter string, as below, is to indicate to the compiler that all the nodes in graph g1 are of type string.

```
graph<string> g1;
g1 = new graph<string>();
```

To add nodes to a graph, we use the ~+ operator, providing it with the graph variable name and the string value of the node's name field.

```
g1~+a1;
g1~+a2;
```

In order to declare a node, we need to specify the node's type parameter. To declare a node of type parameter string, as below, is to indicate to the compiler that node n1's data field is of type string.

We can assign n1 by performing a node retrieval operation on graph g1 with the ~_ graph operator, providing it with the graph variable name and the string value of the node's name field.

```
node<string> n1;
node<string> n2;
n1 = g1~_a1;
n2 = g1~_a2;
```

We can also assign a node by calling the node constructor, providing it with the string value of the node's name field and the type of its data field.

```
node<string> n1;
node<string> n2;
n1 = new node<string>(a1);
n2 = new node<string>(a2);
```

We now want to add a directed edge of edge weight 2 from n1 to n2, and print the resultant graph.

```
g1[2]->(n1,n2);
g1.printGraph();
```

2.2.4 Putting it all Together

The **example.yg** file should now look like this:

```
int main() {

      string a1;
      string a2;
      a1 = "a1";
      a2 = "a2";

      graph<string> g1;
      g1 = new graph<string>();
      g1~+a1;
      g1~+a2;

      node<string> n1;
      n1 = g1~_a1;

      node<string> n2;
      n2 = g1~_a2;

      g1[2]->(n1,n2);

      g1.printGraph();

      return 0;
}
```

The output of the executable should look like this:

```
graph:
node name:  a1
node visited:  0
node inNodes:  {}
node outNodes:  {(a2, 2), }

node name:  a2
node visited:  0
node inNodes:  {(a1, 2), }
node outNodes:  {}
```

3 Language Reference Manual

This language reference manual describes in-depth the lexical conventions, data types, scoping rules, and grammar of yeezyGraph, which is a statically-typed imperative language.

3.1 Lexical Conventions

3.1.1 Identifiers

An identifier refers to a name that is given to entities such as variables, functions, structures etc. In yeezyGraph, identifiers must be unique and may not be double-declared.

Each identifier can be composed of letters, digits, and the underscore character, and must begin with either a letter or an underscore character. Upper and lowercase letters are distinct because yeezyGraph is case-sensitive.

3.1.2 Keywords

Keywords are predefined, reserved words that have special meanings to the yeezyGraph compiler. Keywords are part of the syntax and they cannot be used as identifiers.

3.1.2.1 Statements, Blocks and Control Flow

```
main return if else for while
```

3.1.2.2 Types

```
int float bool string true false
struct list queue pqueue
node graph
```

3.1.2.3 Built-in Functions

```
print printfloat prints printb printstring printint
printGraph printlist l_add l_delete l_get lsize qadd
qfront p_push p_delete p_size qremove qadd qfront size
isEmpty weight contains setData modifyVisited
```

3.1.3 Literals

yeezyGraph supports integer, float, boolean, as well as string literals.

3.1.3.1 Integer Literals

yeezyGraph's integer literals are sequences of one or more decimal digits that have the following regular expression:

```
digit = ['0'–'9']
INT_LITERAL = digit+
```

3.1.3.2 Float Literals

yeezyGraph's float literals are sequences of one or more decimal digits and a single decimal that have the following regular expression:

```
digit = ['0'–'9']
FLOAT_LIT = digit+('.')digit+
```

3.1.3.3 Boolean Literals

yeezyGraph's boolean literals can only take on true or false values. They have the following regular expression:

```
BOOL_LIT = “true” | “false”
```

3.1.3.4 String Literals

yeezyGraph's string literals are null-terminated sequences of characters from the source character set enclosed in double quotation marks (" "). They have the following regular expression:

```
STR_LIT = '"'([^'"']*)'"'
```

3.1.4 Operators

Operator Type	Operators
Arithmetic Operators	+ − * /
Relational Operators	== != < <= > >=
Logical Operators	! && ||
Assignment Operator	=
Struct Access Operator	~
Node Operators	~+ ~_ –> !–>
Graph Operators	@

3.1.4.1 Arithmetic Operators

yeezyGraph supports the 4 basic arithmetic operators: addition, subtraction, multiplication, and division. The two operands of a binary arithmetic operation must be of the same type, which is also the type of the operation's return value.

Multiplication and division have a high operator precedence than addition and subtraction. All arithmetic operators are left-associative.

3.1.4.2 Relational Operators

Relational operators compare the values of the two operands, which must be of the same type. Relational operators return boolean values.

All relational operators are left-associative.

3.1.4.3 Logical Operators

Logical operators are used with boolean values, and return a boolean value.

All logical operators are left-associative.

3.1.4.4 Assignment Operator

The assignment operator takes the value of its right operand and stores it in the left operand.

The assignment operator is right-associative.

3.1.4.5 Struct Access Operator

The struct access operator allows for the access of members in a struct.

The struct access operator is left-associative.

3.1.4.6 Node and Graph Operators

Node and graph operators allow for the access and manipulation of node and graph objects.

All node and graph operators are left-associative.

3.1.4.7 Operator Precedence

Operator precedence determines the order in which operators are evaluated. Operators with higher precedence are evaluated first.

In decreasing order of precedence:

- Node and graph operators
- Struct access operator
- Arithmetic operators: multiplication and division
- Arithmetic operators: addition and subtraction
- Relational operators
- Logical operators
- Assignment operator

3.1.5 Punctuators

Some characters in yeezyGraph are used as punctuators, which have their own syntactic and semantic significance. Punctuators are not operators or identifiers.

3.1.5.1 Semi-colon

The semi-colon is used as a statement terminator in yeezyGraph. It is also used in the syntax of a for loop (see Section 3.3.3.4).

3.1.5.2 Comma

The comma is used to separate the elements of a function argument list or the variables in a data declaration.

3.1.5.3 Parentheses

Parentheses group expressions (when asserting precedence), isolate conditional expressions, and also indicate function calls and function parameters.

3.1.5.4 Curly Braces

Curly braces indicate the start and end of a compound statement, which may have its own local variables. Curly braces are also in struct declarations and function definitions.

3.1.5.5 Angle Brackets

Angle brackets delimit the type parameters in yeezyGraph's generic collection classes (see Section 3.2.2).

3.1.6 Comments

yeezyGraph supports single-line and multi-line comments that begin with /* and terminate with */. Multi-line comments in yeezyGraph do not nest.

3.2 Data Types

There are four categories of data types supported by yeezyGraph, namely:

Type	Data Types
Primitive Data Types	`int float bool string`
Derived Data Types	`struct list queue pqueue`
Node and Graph Data Types	`node graph`
Void Data Type	`void`

3.2.1 Primitive Data Types

Primitive data types – int, float, bool, and string – are the basic building blocks in yeezyGraph. They allow for more complicated composite types to be recursively constructed starting from these basic primitive types.

3.2.1.1 int

`int` is the keyword that designates the 32-bit signed integer primitive type in yeezyGraph. Integers can take any value from the range $-(2^{31})$ to $(2^{31}) - 1$. Integer variables may also be assigned to INFINITY.

Arithmetic operators may be applied to integer variables. To print an integer, use the built-in `print` function.

```
int x;
x = 32;
x = x + 8;
print(x); /* prints 40 */
x = INFINITY;
```

3.2.1.2 float

`float` is the keyword that designates the 32-bit (single-precision) float primitive type in yeezyGraph.

Arithmetic operators may be applied to float variables. To print a float, use the built-in `printfloat` function.

```
float x;
x = 32.0;
x = x + 8.0;
printfloat(x); /* prints 40.0 */
```

3.2.1.3 bool

`bool` is the keyword that designates the boolean primitive type in yeezyGraph. There are only two possible boolean values: true and false.

Logical operators may be applied to boolean variables. To print a bool, use the built-in `print` function.

```
bool x;
bool y;
x = true;
y = x || false;
print(y); /* prints true */
```

3.2.1.4 string

`string` is the keyword that designates the string primitive type in yeezyGraph, representing null-terminated character strings. To print a string, use the built-in `prints` function.

```
string s;
s = “This is a string literal”;
prints(s); /* prints “This is a string literal” */
```

3.2.2 Derived Data Types

Derived data types – struct, list, queue, pqueue – are object types that are aggregates of one or more types of primitive data types.

3.2.2.1 struct

A `struct` is a user-defined composite data type that defines a physically-grouped list of variables that are placed under one name in a contiguous block of memory.

Structs are declared at the global level with the following syntax:

```
struct S1 {
    int x;
    string y;
}
```

Each struct can contain many different primitive and derived data types (excluding other structs).

```
struct S2 {
    struct S1; /* not allowed */
}
```

Struct declarations cannot be nested.

```
struct S2 {

    /* not allowed */
    struct S1 {
      int x;
      string y;
    }

}
```

In order to access any member of a struct, we use the struct access operator **~**.

```
struct S1 {
    int x;
    string y;
}

int main() {
    struct S1 s1;
    s1~x = 32;
    s1~y = "hello world";
    return 0;
}
```

3.2.2.2 list

A `list` is an ordered collection of elements of the same type. Lists may contain duplicate elements.

When declaring a list, the list's type parameter must be specified. Lists are declared with the following syntax:

```
/* initializing an empty list */
list<string> l1;
l1 = new list<string>();

/* initializing a list with elements */
list<int> l2;
l2 = new list<int>(1,2,3);
```

The following built-in functions provide methods to efficiently access, search, and manipulate lists of type E:

Built-in Function	Description
`l_add(E e)`	Appends the specified element to the end of this list, and returns the list
`l_delete(E e)`	Removes the first occurrence of the specified element from this list (if it is present), and returns the list
`l_get(int index)`	Returns the element at the specified position in this list
`l_size()`	Returns the number of elements in this list

The following program demonstrates the usage of the above built-in functions on an integer list, `a`:

```
int main() {

    list<int> a;

    a = new list<int>(1,2,3);
    a.l_add(4);
    a.l_delete(0);

    int x;
    x = a.l_get(0);
    print(x); /* prints 2 */
    return 0;

}
```

3.2.2.3 Queue

A queue is an ordered collection of elements of the same type and follows the FIFO principle. Users are limited to inserting elements at the end of queue and deleting elements from the start of queue. Queues may contain duplicate elements.

When declaring a queue, the queue's type parameter must be specified. Queues are declared with the following syntax:

```
/* initializing an empty queue */
Queue<float> q1;
q1 = new Queue<float>();

/* initializing a queue with elements */
queue<int> q2;
q2 = new queue<int>(1,2,3);
```

The following built-in functions provide methods to efficiently access and manipulate queues of type E:

Built-in Function	Description
`qadd(E e)`	Inserts the specified element into this queue, and returns the queue
`qremove()`	Removes the element at the head of this queue, and returns the queue
`qfront()`	Returns, but does not remove, the head of this queue

The following program demonstrates the usage of the above built-in functions on a float queue, `a`:

```
int main() {

  Queue<float> a;
  a = new Queue<float>();
  a.qadd(3.1);
  a.qadd(5.8);
  x = a.qfront();
  printfloat(x); /* prints 3.1 */

  q.qremove();
  y = a.qfront();
  printfloat(x); /* prints 5.8 */

  return 0;
}
```

3.2.2.4 Priority Queue

A `pqueue`, or priority queue, is an ordered collection of nodes. The node elements of a priority queue are ordered according to the natural ordering of their `data` field, with the head of the pqueue being the *least* element with respect to the specified ordering.

`pqueues` may contain duplicate elements. They are declared with the following syntax:

```
/* initializing an empty pqueue */
pqueue p;
p = new pqueue(n1); /* n1 is an initialized node */
```

The following built-in functions provide methods to efficiently access and manipulate pqueues:

Built-in Function	Description
`p_push(Node n)`	Inserts the specified node into the pqueue, and returns the pqueue
`p_delete()`	Deletes the node at the head of the pqueue, and returns the pqueue
`p_size()`	Returns the number of elements in the pqueue

The following program demonstrates the usage of the above built-in functions on a pqueue:

```
int main() {

    graph<int> g1;
    g1 = new graph<int>();

    g1~+"a";
    g1~+"b";
    g1~+"c";

    node<int> n1;
    n1 = g1~_"a";
    n1.setData(3);

    node<int> n2;
    n2 = g1~_"b";
    n2.setData(1);

    node<int> n3;
    n3 = g1~_"c";
    n3.setData(3);
```

```
    pqueue p;
    p = new pqueue(n1);
    p.p_push(n2);
    p.p_delete(); /* deletes and returns node n2 */

    return 0;

}
```

3.2.3 Node and Graph Data Types

Nodes and graphs are the fundamental data types in yeezyGraph. Their ease of use underpins the language's utility.

3.2.3.1 Graph Data Type

A graph is a collection of nodes *of the same type*: all nodes in a graph must have the same `data` field type (see Section 3.2.3.2 for more details).

When declaring a graph, the graph's type parameter – which is the type of its nodes' `data` field – must be specified. Graphs are declared with the following syntax:

```
/* all nodes in g1 have data of type int */
graph<int> g1;
g1 = new graph<int>();
```

In order to add a node to a graph, we use the **~+** graph operator with the graph variable name, as well as the node's name field (of string type – see Section 3.2.3.2 for more details).

In order to retrieve a node from a graph, we use the **~_** graph operator with the graph variable name, as well as the node's name field (of string type – see Section 3.2.3.2 for more details).

```
string my_name = "Amy";

/* to g1, add node with name: "Amy" */
g1~+my_name;

/* retrieve node with name: "Amy" from graph g1 */
node<int> n1; /* declare a node */
n1 = g1~_(my_name);
```

In order to add a directed edge between two nodes in a graph, we use the **–>** operator. In `g1[x]–>(n1, n2)`, `n1` and `n2` are nodes belonging to graph `g1`, and `x` is the edge weight of the edge (from `n1` to `n2`) that we desire to add.

In order to remove a directed edge between two nodes in a graph, we use the **!–>** operator. In `g1[x]!–>(n1, n2)`, `n1` and `n2` are nodes belonging to graph `g1`, and `x` is the edge weight of the edge (from `n1` to `n2`) that we desire to remove.

```
/* to g1, add another node with a1 = name: “Anne” */
g1~+a1;

/* retrieve node with a1 = name: “Anne” from graph g1 */
node<int> n2; /* declare a node */
n2 = g1~_a1;

/* add a directed edge of weight 2 from n1 to n2 */
g1[2]–>(n1, n2);
g1.printGraph();

/* remove the directed edge of weight 2 from n1 to n2 */
g1[2]!–>(n1, n2);
g1.printGraph();
```

If the graph is undirected and there is an edge between nodes n1 and n2 in graph g1, simply add 2 directed edges between the nodes.

```
/* add an undirected edge of weight 2 from n1 to n2 */
g1[2]–>(n1, n2);
g1[2]–>(n2, n1);
```

The following built-in functions provide methods to efficiently access and manipulate graphs:

Built-in Functions	Description
`printGraph()`	Prints the specified graph and returns void
`size()`	Returns the number of nodes in the specified graph
`isEmpty()`	Returns true if there are no nodes in the specified graph, false otherwise
`weight(node n1, node n2)`	Returns the edge weight between nodes n1 and n2 in the specified graph
`contains("name")`	Returns true if the specified graph contains a node with name field “name”

3.2.3.2 Node Data Type

In yeezyGraph, a node is a data type that contains five different fields: **`name`**, **`visited`**, **`inNodes`**, **`outNodes`**, and **`data`**.

- The **`name`** field is of type string. This field represents the name of the node.
- The **`visited`** field is of type bool. Many graph traversal algorithms require information on whether a node has been visited or not. When a node is declared, the field is initialized to false.
- The **`inNodes`** field is of type map<string, int> (internal data structure), and maps any nodes that are connected to our node by an incoming edge, to the corresponding edgeweight
- The **`outNodes`** field is of type map<string, int> (internal data structure), and maps any nodes that are connected to our node by an outgoing edge, to the corresponding edgeweight
- The **`data`** field can be of any type, but must be specified when a node is initially declared. The type parameter of a node is equivalent to the type of its **`data`** field.

When declaring a node, the node's type parameter – which is the type of its **`data`** field – must be specified. Nodes are declared with the following syntax:

```
/* node n2 has type parameter int */
node<int> n2;
```

We can assign a value to node n2 by performing node retrieval from a graph:

```
/* to graph g1, add a node with name a1 = "Anne" */
g1~+a1;

/* retrieve node with name a1 = "Anne" from graph g1 */
node<int> n2; /* declare node n2 */
n2 = g1~_a2; /* assign node n2 */
```

In order to *access* the values of the **name**, **visited** and **data** fields, we use the **@** operator. *Modifying* the **visited** and **data** fields require built-in functions. The **name** is immutable.

```
/* access name field of n2 */
prints(n2@name); /* prints "Anne" */

/* access visited field of n2 */
printb(n2@visited); /* prints 0, which stands for false */

/* modify data field of n2 to 1 */
n2.setData(1);

/* access data field of n2 */
print(n2@data); /* prints 1 */
```

The following built-in functions provide methods to modify the **visited** and **data** fields in nodes:

Built-in Functions	Description
`setData(E e)`	Modifies the **data** field of the specified node to value e. e's type (E) must be the same as the specified node's type.
`modifyVisited(bool b)`	Modifies the **visited** field of the specified node to b.

```
/* modifies visited field of n2 */
n2.modifyVisited(true);
printb(n2@visited); /* prints 0, which stands for false */
```

3.2.4 Void Data Type

The `void` type is used as the result of a function that returns normally, but does not provide a result value to its caller. Functions with `void` return types are usually called for their side effects.

3.3 Program Structure

3.3.1 Program Syntax

Every statement in yeezyGraph must belong to one of three blocks: function declarations, global variable declarations, or struct type declarations. Statements *outside* any function declaration – basically global variable declarations and struct type declarations – are considered to be in the global scope. Conversely, statements *inside* any function declaration are considered to be in the local scope of the function it is enclosed by.

The main() function is the entry point of any yeezyGraph program, where the execution of the program starts. Every yeezyGraph program must have a mandatory main() function, declared once in the source code.

3.3.1.1 Variable Declarations

Variable declarations determine the size and layout of the variable in memory and the set of operations that can be applied to the variable. As yeezyGraph is a statically-typed language, every variable's type is identified by the compiler at compile-time based on its initial variable declaration.

Variables must be declared before they can be assigned. In yeezyGraph, variables are declared by stating the variable type, followed by the variable name. As with all identifiers, variable names can be composed of letters, digits, and the underscore character, and must begin with either a letter or an underscore character. Upper and lowercase letters are distinct because yeezyGraph is case-sensitive. Variable names must be unique.

```
variable_type variable_name;
```

Variables can be declared and initialized globally (in a global variable declaration block), or locally within a function (in a function declaration block).

```
int x; /* global variable declaration */
x = 1;

/* main function declaration */
int main() {
    int y; /* local variable declaration */
    y = 2;
}
```

3.3.1.2 Function Declarations

A function declaration in yeezyGraph consists of a function header and a function body.

```
return_type function_name(parameter list) {
   function_body
}
```

The function signature is composed of the function's return type and name. The function's return type is the data type of the value that the function returns. If no value is returned, the function's return type is `void`. Function names must adhere to identifier name protocol and must also be unique.

The parameter list refers to the type, order, and number of the parameters belonging to a function. Parameters are optional – a function may take no parameters. When a function is called, the function call statement must have the same type, order, and number of parameters as declared in the function declaration.

```
function_name(parameter list)
```

The function body is a collection of statements that define what the function does when called. If a function contains a return statement, it must return a value of return type specified in the function declaration, else the compiler will throw an error.

The mandatory function in yeezyGraph is `main`, which is the starting execution point of any yeezyGraph program. The `main` function takes no parameters.

3.3.1.3 Struct Declarations

A struct is a user-defined composite data type that defines a physically-grouped list of variables that are placed under one name in a contiguous block of memory. Each struct can contain many different primitive and derived data types (excluding other structs). Struct declarations cannot be nested.

```
struct [struct_name] {
   member_definition;
   member_definition;
   ...
   member_definition;
}
```

3.3.2 Expressions

An expression is any legal combination of symbols that represents a value. Legal expressions in yeezyGraph include:

- Literals
  - Integer
  - Float
  - Boolean
  - String
  - Infinity
  - Negative Infinity
- Variable names
- Declarations
  - List
  - Queue
  - Priority Queue
  - Node
  - Graph
- Arithmetic operations
  - Binary
  - Unary
- Node Operations
- Graph Operations
- Struct Operations
- Assignments
- Function calls
- Object function calls
- No expression

3.3.3 Statements

yeezyGraph's programs consist of a series of statements.

3.3.3.1 Expression Statements

An expression statement consists of an expression followed by a semicolon.

3.3.3.2 Block Statements

A block statement groups multiple statements into a single statement. It consists of multiple statements and declarations enclosed within curly braces.

3.3.3.3 If-Else Selection Statements

There are two types of selection statements in yeezyGraph that allow for branching of execution based on a conditional boolean expression.

```
if (expression) statement;
```

In the above type of if-statement, the sub-statement will only be executed if the condition expression evaluates to true.

```
if (expression) statement else statement;
```

In the above type of if-statement, the first statement will only be executed if the condition expression evaluates to true; otherwise, the second statement will be executed. Each else matches up with the closest unmatched if.

3.3.3.4 For Iteration Statements

```
for (expression1; expression2; expression3) statement;
```

In a for loop, `expression1` represents an initial condition. This expression can be optional.

`expression2` represents a control expression. If it evaluates to true, the body of the for loop is executed. If it evaluates to false, the body of the loop does not execute and the program's execution jumps to the next statement just after the for loop.

`expression3` allows the user to update any loop control variables following each iteration of the for loop. This expression can be optional.

3.3.3.5 While Iteration Statements

```
while (expression) statement;
```

In a while statement, the statement body runs repeatedly so long as the control expression evaluates to true at the beginning of each iteration.

3.3.4 Scoping Rules

Every variable in yeezyGraph has a lexical, static scope. A variable declared inside a curly braces-enclosed block is accessible inside the block and all inner blocks nested within that block, but is not accessible outside the block.

If an inner block declares a variable with the same name as a variable declared by the outer enclosing block, then the visibility of the outer block variable ends at the point of the variable's declaration by the inner block.

4 Project Plan

4.1 Planning, Specification, Development, and Testing

Based on the deadlines of the project deliverables that were set by Professor Edwards, we had an initial idea of what the project milestones were. Completing the language proposal and the language reference manual allowed us to better understand the scope of our project, including all necessary features and specifications that we had to implement in our language. We then proceeded to complete the compiler front-end, starting with the scanner, parser, and finally the ast.

From there, we utilized a vertical development process to build up each language feature in the compiler back-end. Each language feature was implemented end-to-end from code generation to semantic checking and finally to unit-testing. Integration testing was also set up so that the newly-added code could pass all new and existing test reviews before the changes were finally commited and pushed to master.

More specific deadlines were set when the yeezyGraph team met for weekly office hours with our TA, Alexandra Medway, who helped us to gauge our project progress and also resolve any outstanding issues that we had encountered in our project over the course of the week. During our second meeting of the week, we then set action items for each team member to complete. Additionally, we tried to troubleshoot coding issues and work out design and implementation details together during our weekly meetings as a team.

4.2 Programming Style Guide

4.2.1 OCaml Style Guide

- Limit lines to 80 characters
- Include line breaks
- No tab characters – indent with 2 spaces
- Indent to indicate nesting and scope
- Include comments, as much as possible, above the code they reference
- Follow the standard OCaml naming conventions (as per the standard OCaml libraries)
- Use meaningful, descriptive names
- Break up large functions into smaller ones

4.2.2 yeezyGraph Style Guide

- Limit lines to 80 characters
- Include line breaks
- No tab characters – indent with 4 spaces
- Indent to indicate nesting and scope
- Include comments, as much as possible, above the code they reference
- Use meaningful, descriptive names
- Use lowercase letters and underscores for variable names
- Use PascalCase for user-defined function names

4.3 Project Timeline

Date	Milestone
February 08	Language proposal complete
February 22	Language reference manual complete
March 17	Scanner, parser, and ast implemented
March 27	Hello world compiles
May 9	Regression testing, debugging complete
May 9	Presentation complete
May 10	Project report complete

4.4 Roles and Responsibilities

All team members were involved with most parts of the project due to our vertical development process.

Team Member	Role
Nancy Xu	System architect
Wanlin Xie	Project manager
Yiming Sun	Language guru

4.5 Software Development Environment

Languages	Ocaml 4.04.1, GCC
Editors	Sublime Text, Vim
Version Control	Git
Operating Systems	Ubuntu 15.04, OS X 10.5

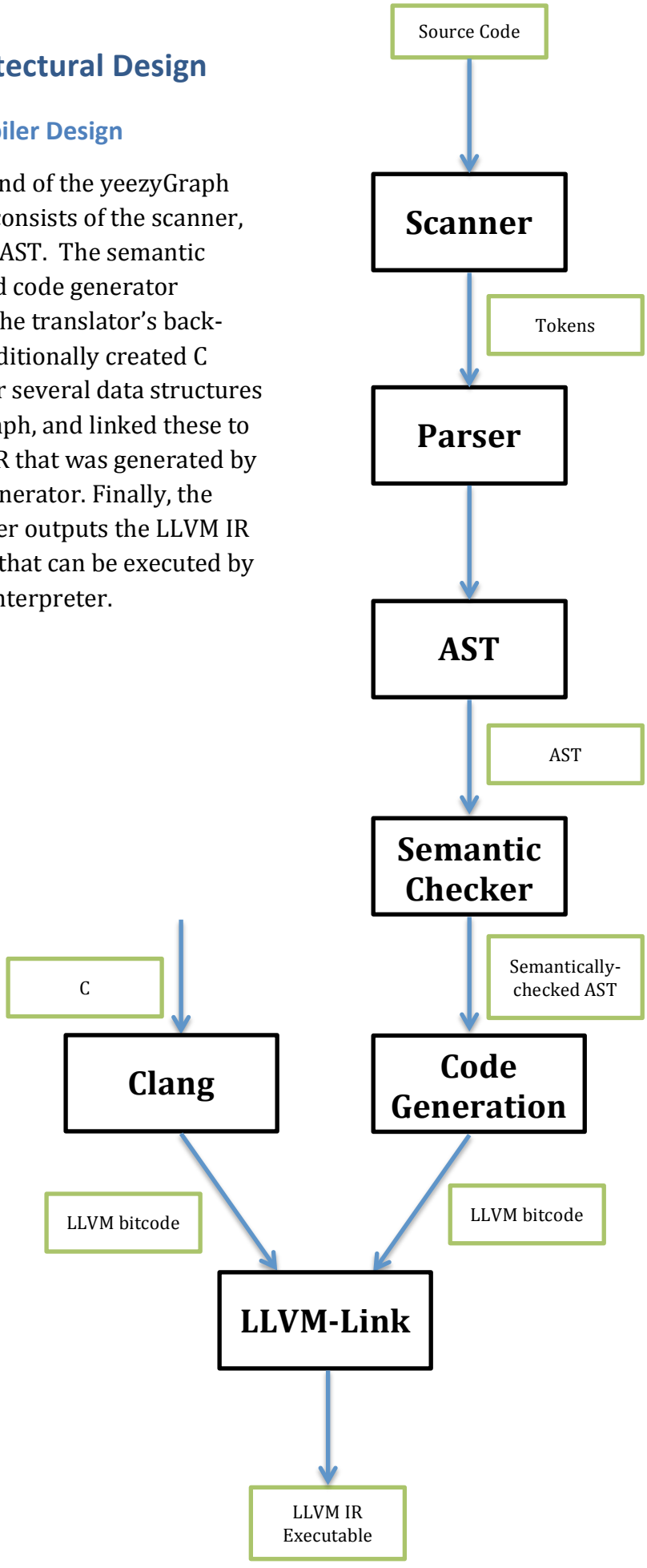
4.6 Project Log

Our project log is in our group github account: https://github.com/kiyun-k/yeezyGraph

5 Architectural Design

5.1 Compiler Design

The front-end of the yeezyGraph translator consists of the scanner, parser and AST. The semantic checker and code generator constitute the translator's back-end. We additionally created C modules for several data structures in yeezyGraph, and linked these to the LLVM IR that was generated by the code generator. Finally, the LLVM-Linker outputs the LLVM IR executable that can be executed by the LLVM interpreter.



5.1.1 Scanner (Nancy, Wanlin, Yiming)

The scanner takes in as input a yeezyGraph source program and generates tokens for identifiers, keywords, operators, and values.

5.1.2 Parser and AST (Nancy, Wanlin, Yiming)

The parser takes in as input the tokens generated from the scanner and constructs an abstract syntax tree (AST) based on the yeezyGraph grammar.

5.1.3 Semantic Checker (Nancy, Wanlin, Yiming)

The semantic checker takes in as input the AST generated by the parser, and recursively checks for semantic errors. If any errors are detected, an exception is raised.

5.1.4 Code Generator (Nancy, Wanlin, Yiming)

The code generator performs post-order traversal on the semantically-checked AST, producing the LLVM IR using the OCaml-LLVM bindings.

6 Test Plan

We performed both unit and integration testing on each feature of our program that we implemented. For each feature, we had two or three tests that confirmed the correct declaration, assignment, and other needed operations upon each feature. As we progressed in implementing each feature of our program, we also combined the testing for several of our features in the same program, in order ensure that all of the components of our language sync together.

Below are a few of the tests we used, along with their produced output.

6.1 Struct Testing

This tests the declaration of structs, the assignment of values to structs, and the printing of values in structs.

```
struct Point {
        int x;
        int y;
}

struct Circle {
        int x;
        int y;
        string name;
}

int main() {
        struct Point p;
        struct Circle c;
        p~x = 5;
        p~y = 10;
        c~x = 1;
        c~y = 2;
        c~name = "circle";
        print(p~x);
        print(c~y);
        prints(c~name);
        return 0;
}
```

Produced output:

```
5
2
circle
```

6.2 Queue Testing

This program tests the declaration and assignment to queues. It also tests the built-in functions qadd and qfront. This is also an integration test, as it tests the functionality of strings alongside queues.

```
int main() {
        string a;
        string b;
        string c;
        string d;
        Queue<string> q;
        q = new Queue<string>();

        /* adding to a queue */
        a = "abc";
        b = "def";
        q.qadd(a);
        q.qadd(b);
        c = q.qfront();
        prints(c);
        q.qremove();
        d = q.qfront();
        prints(d);
        return 0;

}
```

Produced output:

```
abc
def
```

6.3 Graph Testing

This program tests the declaration and assignment for nodes and graphs. It also tests the built in functions for adding nodes, retrieving nodes from graphs, adding edges, and accessing the fields of each node. It also tests the built-in functions for setting the data for the nodes, retrieving the weight of the edge between two nodes, and printing the graph. It is also an integration test, as it tests the functionality of lists alongside the functionality of graphs, since it assigns the retrieved neighbors of a node to a list.

```
int main() {
        graph<string> g1;
        int x;
        string a1;
        string a2;
        string a3;
        string a4;
        int y;
        node<string> n1;
        node<string> n2;
        list<string> l;
        l = new list<string>();
        g1 = new graph<string>();
        a1 = "abc";
        a2 = "def";
        g1~+a1;
        g1~+a2;
        n1 = g1~_a1;
        n1.setData("abc");
        a3 = n1@data;
        prints(a3);
        n2 = g1~_a2;
        g1[2]->(n1,n2);
        x = g1.weight(n1, n2);
        print(x);
        g1.printGraph();
        l = n1@outNodes;
        l.printList();
        y = l.lsize();
        print(y);
        return 0;

}
```

Produced output:

```
abc
2
graph:
node name:  abc
node visited:  0
node inNodes:  {}
node outNodes:  {(def, 2), }

node name:  def
node visited:  0
node inNodes:  {(abc, 2), }
node outNodes:  {}

{def, }
1
```

6.4 Automation

Our compile script is called **./testall.sh**, which is in the main directory. This script compiles all the files in the **test** directory to LLVM code which can be compiled with clang and run.

6.5 Test Suites

Our tests reside in the **test** directory.

We tested the following features of our language:

- Primitive data declaration, assignment, and operations
- Float declaration, assignment, and arithmetic
- Control flow for for loops, while loops
- Struct declaration, assignment, and operations
- Collection (queues, pqueues, lists) declaration, assignment, operations, and built-in functions
- Nodes and graphs declaration, assignment, operations, and built-in functions
- User-defined algorithm functions: BFS, Dijkstra's Algorithm

6.6 Yeezygraph to LLVM

Graphdeclstruct.yg is a source program located in our tests folder:

```
struct nodedata {
        int dist;
        string parent;
}

int main() {

        graph<struct nodedata> g1;
        string a1;
        node<struct nodedata> n1;
        struct nodedata x;
        struct nodedata y;
        x~dist = 5;
        x~parent = "abc";
        g1 = new graph<struct nodedata>();
        a1 = "abc";
        g1~+a1;
        n1 = g1~_a1;
        n1.setData(x);
        y = n1@data;
        print(y~dist);
        return 0;
}
```

Generated LLVM code:

```
; ModuleID = 'MicroC'

%struct.QueueId = type { %struct.Node*, %struct.Node*, i32 }
%struct.Node = type { i8*, %struct.Node* }
%struct.List = type { %struct.ListNode*, i32 }
%struct.ListNode = type { i8*, %struct.ListNode* }
%struct.pqueue = type { %struct.node**, i32, i32 }
%struct.node = type { i8*, i8, %struct.map*, %struct.map*,
i8* }
%struct.map = type { i32, i32, i8**, i32* }
%struct.graph = type { %struct.List.5*, i32 }
%struct.List.5 = type { %struct.ListNode.4*, i32 }
%struct.ListNode.4 = type { i8*, %struct.ListNode.4* }
%nodedata = type <{ i8*, i32 }>

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.2 = private unnamed_addr constant [3 x i8] c"%d\00"
@fmt.3 = private unnamed_addr constant [3 x i8] c"%s\00"
@fmt.4 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@str = private unnamed_addr constant [4 x i8] c"abc\00"
@str.5 = private unnamed_addr constant [4 x i8] c"abc\00"

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

declare %struct.QueueId* @initQueueId()

declare void @enqueue(%struct.QueueId*, i8*)

declare void @dequeue(%struct.QueueId*)

declare i8* @front(%struct.QueueId*)

declare i32 @q_size(%struct.QueueId*)

declare %struct.List* @l_init()

declare void @l_add(%struct.List*, i8*)

declare void @l_delete(%struct.List*, i32)

declare i8* @l_get(%struct.List*, i32)

declare i32 @l_size(%struct.List*)

declare void @print_list(%struct.List*)

declare %struct.pqueue* @pq_init()

declare void @pq_push(%struct.pqueue*, %struct.node*)

declare %struct.node* @pq_delete(%struct.pqueue*)

declare i32 @p_size(%struct.pqueue*)
```

```
declare void @pq_push(%struct.pqueue*, %struct.node*)

declare %struct.node* @pq_delete(%struct.pqueue*)

declare i32 @p_size(%struct.pqueue*)

declare %struct.node* @n_init(i8*)

declare void @set_data(%struct.node*, i8*)

declare i8* @get_name(%struct.node*)

declare i1 @get_visited(%struct.node*)

declare void @modify_visited(%struct.node*, i1)

declare i8* @get_data(%struct.node*)

declare %struct.List* @get_inNodes(%struct.node*)

declare %struct.List* @get_outNodes(%struct.node*)

declare %struct.graph* @g_init()

declare void @addNode(%struct.graph*, %struct.node*)

declare void @removeNode(%struct.graph*, %struct.node*)

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

declare void @removeEdge(%struct.graph*, %struct.node*,
%struct.node*)

declare void @printGraph(%struct.graph*)

declare i1 @isEmpty(%struct.graph*)

declare i32 @size(%struct.graph*)

declare i1 @contains(%struct.graph*, i8*)

declare %struct.node* @getNode(%struct.graph*, i8*)

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

define i32 @main() {
entry:
  %g1 = alloca %struct.graph*
  %a1 = alloca i8*
  %n1 = alloca %struct.node*
  %x = alloca %nodedata
  %y = alloca %nodedata
```

6.7 Testing Roles

Yiming was in charge of designing, writing, and debugging the unit and integration tests for strings and structs. Similarly, Wanlin's domain was lists and pqueues, and Nancy's domain was floats, queues, nodes, and graphs.

6.8 Dijkstra's Algorithm: Source Language

6.8.1 dijkstra.yg

Shown below is only Dijkstra's algorithm. In the file, the graph to be analyzed is built in the main method.

```
void dijkstra(graph<struct nodedata> g1, node<struct nodedata>
source) {
        pqueue p;
        struct nodedata sourcedata;
        struct nodedata sourcedata2;
        node <struct nodedata> temp;
        node <struct nodedata> temp2;
        string tmpstring;
        list<string> listtemp;
        int i;

        listtemp = new list<string>();
        p = new pqueue();
        tmpstring = "temporary string";

        /* set distance of source vertex to 0 */
        sourcedata = source@data;
        sourcedata~dist = 0;
        source.setData(sourcedata);
        p.p_push(source);

        prints("Vertex              Distance from source");
        while(p.p_size() != 0) {
                temp = p.p_delete();
                printstring(temp@name);
                printstring("                         ");
                sourcedata = temp@data;
                printint(sourcedata~dist);
                listtemp = temp@outNodes;
                print(listtemp.lsize());
                for (i = 0; i < listtemp.lsize(); i = i + 1) {
                        tmpstring = listtemp.l_get(i);
                        temp2 = g1~_tmpstring;
                        sourcedata2 = temp2@data;
                        if (sourcedata2~dist > sourcedata~dist
                              + g1.weight(temp, temp2)) {
                                sourcedata2~dist =
                                sourcedata~dist +
                                g1.weight(temp, temp2);
                                temp2.setData(sourcedata2);
                                p.p_push(temp2);
                        }
                }
        }
```

6.8.2 Dijkstra's Algorithm: LLVM

```
; ModuleID = 'MicroC'

%struct.QueueId = type { %struct.Node*, %struct.Node*, i32 }
%struct.Node = type { i8*, %struct.Node* }
%struct.List = type { %struct.ListNode*, i32 }
%struct.ListNode = type { i8*, %struct.ListNode* }
%struct.pqueue = type { %struct.node**, i32, i32 }
%struct.node = type { i8*, i8, %struct.map*, %struct.map*, i8*
}
%struct.map = type { i32, i32, i8**, i32* }
%struct.graph = type { %struct.List.5*, i32 }
%struct.List.5 = type { %struct.ListNode.4*, i32 }
%struct.ListNode.4 = type { i8*, %struct.ListNode.4* }
%nodedata = type <{ %struct.node*, i32 }>

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.2 = private unnamed_addr constant [3 x i8] c"%d\00"
@fmt.3 = private unnamed_addr constant [3 x i8] c"%s\00"
@fmt.4 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@str = private unnamed_addr constant [2 x i8] c"a\00"
@str.5 = private unnamed_addr constant [2 x i8] c"b\00"
@str.6 = private unnamed_addr constant [2 x i8] c"c\00"
@str.7 = private unnamed_addr constant [2 x i8] c"d\00"
@fmt.8 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.9 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.10 = private unnamed_addr constant [3 x i8] c"%d\00"
@fmt.11 = private unnamed_addr constant [3 x i8] c"%s\00"
@fmt.12 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@str.13 = private unnamed_addr constant [17 x i8] c"temporary
string\00"
@str.14 = private unnamed_addr constant [41 x i8] c"Vertex
Distance from source\00"
@str.15 = private unnamed_addr constant [24 x i8] c"
\00"

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

declare %struct.QueueId* @initQueueId()

declare void @enqueue(%struct.QueueId*, i8*)

declare void @dequeue(%struct.QueueId*)

declare i8* @front(%struct.QueueId*)

declare i32 @q_size(%struct.QueueId*)

declare %struct.List* @l_init()

declare void @l_add(%struct.List*, i8*)

declare void @l_delete(%struct.List*, i32)
```

```
declare %struct.node* @pq_delete(%struct.pqueue*)

declare i32 @p_size(%struct.pqueue*)

declare %struct.node* @n_init(i8*)

declare void @set_data(%struct.node*, i8*)

declare i8* @get_name(%struct.node*)

declare i1 @get_visited(%struct.node*)

declare void @modify_visited(%struct.node*, i1)

declare i8* @get_data(%struct.node*)

declare %struct.List* @get_inNodes(%struct.node*)

declare %struct.List* @get_outNodes(%struct.node*)

declare %struct.graph* @g_init()

declare void @addNode(%struct.graph*, %struct.node*)

declare void @removeNode(%struct.graph*, %struct.node*)

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

declare void @removeEdge(%struct.graph*, %struct.node*,
%struct.node*)

declare void @printGraph(%struct.graph*)

declare i1 @isEmpty(%struct.graph*)

declare i32 @size(%struct.graph*)

declare i1 @contains(%struct.graph*, i8*)

declare %struct.node* @getNode(%struct.graph*, i8*)

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

define i32 @main() {
entry:
  %listtemp = alloca %struct.List*
  %listtemp2 = alloca %struct.List*
  %g1 = alloca %struct.graph*
  %a1 = alloca i8*
  %a2 = alloca i8*
  %a3 = alloca i8*
  %a4 = alloca i8*
```

```
%stringtmp = alloca i8*
  %i = alloca i32
  %n1 = alloca %nodedata
  %n2 = alloca %nodedata
  %n3 = alloca %nodedata
  %n4 = alloca %nodedata
  %a = alloca %struct.node*
  %b = alloca %struct.node*
  %c = alloca %struct.node*
  %d = alloca %struct.node*
  %struct_field_pointer = getelementptr inbounds %nodedata,
%nodedata* %n1, i32 0, i32 1
  store i32 1000000, i32* %struct_field_pointer
  %struct_field_pointer1 = getelementptr inbounds %nodedata,
%nodedata* %n2, i32 0, i32 1
  store i32 1000000, i32* %struct_field_pointer1
  %struct_field_pointer2 = getelementptr inbounds %nodedata,
%nodedata* %n3, i32 0, i32 1
  store i32 1000000, i32* %struct_field_pointer2
  %struct_field_pointer3 = getelementptr inbounds %nodedata,
%nodedata* %n4, i32 0, i32 1
  store i32 1000000, i32* %struct_field_pointer3
  store i8* getelementptr inbounds ([2 x i8], [2 x i8]* @str,
i32 0, i32 0), i8** %a1
  store i8* getelementptr inbounds ([2 x i8], [2 x i8]* @str.5,
i32 0, i32 0), i8** %a2
  store i8* getelementptr inbounds ([2 x i8], [2 x i8]* @str.6,
i32 0, i32 0), i8** %a3
  store i8* getelementptr inbounds ([2 x i8], [2 x i8]* @str.7,
i32 0, i32 0), i8** %a4
  %init = call %struct.graph* @g_init()
  store %struct.graph* %init, %struct.graph** %g1
  %char_n_val_pointer = load i8*, i8** %a1
  %init4 = call %struct.node* @n_init(i8* %char_n_val_pointer)
  %node_alloca = alloca %struct.node*
  store %struct.node* %init4, %struct.node** %node_alloca
  %graph_pointer = load %struct.graph*, %struct.graph** %g1
  %node_pointer = load %struct.node*, %struct.node**
%node_alloca
  call void @addNode(%struct.graph* %graph_pointer,
%struct.node* %node_pointer)
  %char_n_val_pointer5 = load i8*, i8** %a2
  %init6 = call %struct.node* @n_init(i8* %char_n_val_pointer5)
  %node_alloca7 = alloca %struct.node*
store %struct.node* %init6, %struct.node** %node_alloca7
  %graph_pointer8 = load %struct.graph*, %struct.graph** %g1
  %node_pointer9 = load %struct.node*, %struct.node**
%node_alloca7
  call void @addNode(%struct.graph* %graph_pointer8,
%struct.node* %node_pointer9)
```

```
%char_n_val_pointer10 = load i8*, i8** %a3
  %init11 = call %struct.node* @n_init(i8*
%char_n_val_pointer10)
  %node_alloca12 = alloca %struct.node*
  store %struct.node* %init11, %struct.node** %node_alloca12
  %graph_pointer13 = load %struct.graph*, %struct.graph** %g1
  %node_pointer14 = load %struct.node*, %struct.node**
%node_alloca12
  call void @addNode(%struct.graph* %graph_pointer13,
%struct.node* %node_pointer14)
  %char_n_val_pointer15 = load i8*, i8** %a4
  %init16 = call %struct.node* @n_init(i8*
%char_n_val_pointer15)
  %node_alloca17 = alloca %struct.node*
  store %struct.node* %init16, %struct.node** %node_alloca17
  %graph_pointer18 = load %struct.graph*, %struct.graph** %g1
  %node_pointer19 = load %struct.node*, %struct.node**
%node_alloca17
  call void @addNode(%struct.graph* %graph_pointer18,
%struct.node* %node_pointer19)
  %g120 = load %struct.graph*, %struct.graph** %g1
  %char_n_val_pointer21 = load i8*, i8** %a1
  %0 = call %struct.node* @getNode(%struct.graph* %g120, i8*
%char_n_val_pointer21)
  store %struct.node* %0, %struct.node** %a
  %g122 = load %struct.graph*, %struct.graph** %g1
  %char_n_val_pointer23 = load i8*, i8** %a2
  %1 = call %struct.node* @getNode(%struct.graph* %g122, i8*
%char_n_val_pointer23)
  store %struct.node* %1, %struct.node** %b
  %g124 = load %struct.graph*, %struct.graph** %g1
  %char_n_val_pointer25 = load i8*, i8** %a3
  %2 = call %struct.node* @getNode(%struct.graph* %g124, i8*
%char_n_val_pointer25)
  store %struct.node* %2, %struct.node** %c
  %g126 = load %struct.graph*, %struct.graph** %g1
  %char_n_val_pointer27 = load i8*, i8** %a4
  %3 = call %struct.node* @getNode(%struct.graph* %g126, i8*
%char_n_val_pointer27)
  store %struct.node* %3, %struct.node** %d
  %a28 = load %struct.node*, %struct.node** %a
  %n129 = load %nodedata, %nodedata* %n1
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%nodedata*
getelementptr (%nodedata, %nodedata* null, i32 1) to i32))
  %tmp = bitcast i8* %malloccall to %nodedata*
  store %nodedata %n129, %nodedata* %tmp
  %ptr = bitcast %nodedata* %tmp to i8*
  call void @set_data(%struct.node* %a28, i8* %ptr)
  %g130 = load %struct.graph*, %struct.graph** %g1
  %e1_val_pointer = load %struct.node*, %struct.node** %a
  %e2_val_pointer = load %struct.node*, %struct.node** %b
  call void @addEdge(%struct.graph* %g130, %struct.node*
%e1_val_pointer, %struct.node* %e2_val_pointer, i32 1)
```

```
%g131 = load %struct.graph*, %struct.graph** %g1
  %e1_val_pointer32 = load %struct.node*, %struct.node** %a
  %e2_val_pointer33 = load %struct.node*, %struct.node** %c
  call void @addEdge(%struct.graph* %g131, %struct.node*
%e1_val_pointer32, %struct.node* %e2_val_pointer33, i32 2)
  %b34 = load %struct.node*, %struct.node** %b
  %n235 = load %nodedata, %nodedata* %n2
  %malloccall.36 = tail call i8* @malloc(i32 ptrtoint
(%nodedata* getelementptr (%nodedata, %nodedata* null, i32 1)
to i32))
  %tmp37 = bitcast i8* %malloccall.36 to %nodedata*
  store %nodedata %n235, %nodedata* %tmp37
  %ptr38 = bitcast %nodedata* %tmp37 to i8*
  call void @set_data(%struct.node* %b34, i8* %ptr38)
  %g139 = load %struct.graph*, %struct.graph** %g1
  %e1_val_pointer40 = load %struct.node*, %struct.node** %b
  %e2_val_pointer41 = load %struct.node*, %struct.node** %a
  call void @addEdge(%struct.graph* %g139, %struct.node*
%e1_val_pointer40, %struct.node* %e2_val_pointer41, i32 1)
  %g142 = load %struct.graph*, %struct.graph** %g1
  %e1_val_pointer43 = load %struct.node*, %struct.node** %b
  %e2_val_pointer44 = load %struct.node*, %struct.node** %d
  call void @addEdge(%struct.graph* %g142, %struct.node*
%e1_val_pointer43, %struct.node* %e2_val_pointer44, i32 10)
  %c45 = load %struct.node*, %struct.node** %c
  %n346 = load %nodedata, %nodedata* %n3
  %malloccall.47 = tail call i8* @malloc(i32 ptrtoint
(%nodedata* getelementptr (%nodedata, %nodedata* null, i32 1)
to i32))
  %tmp48 = bitcast i8* %malloccall.47 to %nodedata*
  store %nodedata %n346, %nodedata* %tmp48
  %ptr49 = bitcast %nodedata* %tmp48 to i8*
  call void @set_data(%struct.node* %c45, i8* %ptr49)
  %g150 = load %struct.graph*, %struct.graph** %g1
  %e1_val_pointer51 = load %struct.node*, %struct.node** %c
  %e2_val_pointer52 = load %struct.node*, %struct.node** %a
  call void @addEdge(%struct.graph* %g150, %struct.node*
%e1_val_pointer51, %struct.node* %e2_val_pointer52, i32 2)
  %g153 = load %struct.graph*, %struct.graph** %g1
  %e1_val_pointer54 = load %struct.node*, %struct.node** %c
  %e2_val_pointer55 = load %struct.node*, %struct.node** %d
  call void @addEdge(%struct.graph* %g153, %struct.node*
%e1_val_pointer54, %struct.node* %e2_val_pointer55, i32 5)
  %d56 = load %struct.node*, %struct.node** %d
  %n457 = load %nodedata, %nodedata* %n4
  %malloccall.58 = tail call i8* @malloc(i32 ptrtoint
(%nodedata* getelementptr (%nodedata, %nodedata* null, i32 1)
to i32))
  %tmp59 = bitcast i8* %malloccall.58 to %nodedata*
  store %nodedata %n457, %nodedata* %tmp59
  %ptr60 = bitcast %nodedata* %tmp59 to i8*
  call void @set_data(%struct.node* %d56, i8* %ptr60)
  %g161 = load %struct.graph*, %struct.graph** %g1
  %e1_val_pointer62 = load %struct.node*, %struct.node** %d
  %e2_val_pointer63 = load %struct.node*, %struct.node** %b
```

```
call void @addEdge(%struct.graph* %g161, %struct.node*
%e1_val_pointer62, %struct.node* %e2_val_pointer63, i32 10)
  %g164 = load %struct.graph*, %struct.graph** %g1
  %e1_val_pointer65 = load %struct.node*, %struct.node** %d
  %e2_val_pointer66 = load %struct.node*, %struct.node** %c
  call void @addEdge(%struct.graph* %g164, %struct.node*
%e1_val_pointer65, %struct.node* %e2_val_pointer66, i32 5)
  %a67 = load %struct.node*, %struct.node** %a
  %g168 = load %struct.graph*, %struct.graph** %g1
  call void @dijkstra(%struct.graph* %g168, %struct.node* %a67)
  ret i32 0
}

define void @dijkstra(%struct.graph* %g1, %struct.node*
%source) {
entry:
  %g11 = alloca %struct.graph*
  store %struct.graph* %g1, %struct.graph** %g11
  %source2 = alloca %struct.node*
  store %struct.node* %source, %struct.node** %source2
  %p = alloca %struct.pqueue*
  %sourcedata = alloca %nodedata
  %sourcedata2 = alloca %nodedata
  %temp = alloca %struct.node*
  %temp2 = alloca %struct.node*
  %tmpstring = alloca i8*
  %listtemp = alloca %struct.List*
  %i = alloca i32
  %init = call %struct.List* @l_init()
  store %struct.List* %init, %struct.List** %listtemp
  %init3 = call %struct.pqueue* @pq_init()
  store %struct.pqueue* %init3, %struct.pqueue** %p
  store i8* getelementptr inbounds ([17 x i8], [17 x i8]*
@str.13, i32 0, i32 0), i8** %tmpstring
  %source4 = load %struct.node*, %struct.node** %source2
  %getData = call i8* @get_data(%struct.node* %source4)
  %d_ptr = bitcast i8* %getData to %nodedata*
  %d_ptr5 = load %nodedata, %nodedata* %d_ptr
  store %nodedata %d_ptr5, %nodedata* %sourcedata
  %struct_field_pointer = getelementptr inbounds %nodedata,
%nodedata* %sourcedata, i32 0, i32 1
  store i32 0, i32* %struct_field_pointer
  %source6 = load %struct.node*, %struct.node** %source2
  %sourcedata7 = load %nodedata, %nodedata* %sourcedata
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%nodedata*
getelementptr (%nodedata, %nodedata* null, i32 1) to i32))
  %tmp = bitcast i8* %malloccall to %nodedata*
  store %nodedata %sourcedata7, %nodedata* %tmp
  %ptr = bitcast %nodedata* %tmp to i8*
  call void @set_data(%struct.node* %source6, i8* %ptr)
%p8 = load %struct.pqueue*, %struct.pqueue** %p
  %source9 = load %struct.node*, %struct.node** %source2
  call void @pq_push(%struct.pqueue* %p8, %struct.node*
%source9)
```

```
%printf = call i32 (i8*, ...) @printf(i8* getelementptr
inbounds ([4 x i8], [4 x i8]* @fmt.9, i32 0, i32 0), i8*
getelementptr inbounds ([41 x i8], [41 x i8]* @str.14, i32 0,
i32 0))
  br label %while

while:                                            ; preds =
%merge61, %entry
  %p62 = load %struct.pqueue*, %struct.pqueue** %p
  %isEmpty = call i32 @p_size(%struct.pqueue* %p62)
  %tmp63 = icmp ne i32 %isEmpty, 0
  br i1 %tmp63, label %while_body, label %merge64

while_body:                                       ; preds =
%while
  %p10 = load %struct.pqueue*, %struct.pqueue** %p
  %data = call %struct.node* @pq_delete(%struct.pqueue* %p10)
  store %struct.node* %data, %struct.node** %temp
  %temp11 = load %struct.node*, %struct.node** %temp
  %getName = call i8* @get_name(%struct.node* %temp11)
  %printf12 = call i32 (i8*, ...) @printf(i8* getelementptr
inbounds ([3 x i8], [3 x i8]* @fmt.11, i32 0, i32 0), i8*
%getName)
  %printf13 = call i32 (i8*, ...) @printf(i8* getelementptr
inbounds ([3 x i8], [3 x i8]* @fmt.11, i32 0, i32 0), i8*
getelementptr inbounds ([24 x i8], [24 x i8]* @str.15, i32 0,
i32 0))
  %temp14 = load %struct.node*, %struct.node** %temp
  %getData15 = call i8* @get_data(%struct.node* %temp14)
  %d_ptr16 = bitcast i8* %getData15 to %nodedata*
  %d_ptr17 = load %nodedata, %nodedata* %d_ptr16
  store %nodedata %d_ptr17, %nodedata* %sourcedata
  %struct_field_pointer18 = getelementptr inbounds %nodedata,
%nodedata* %sourcedata, i32 0, i32 1
  %struct_field_value = load i32, i32* %struct_field_pointer18
  %printf19 = call i32 (i8*, ...) @printf(i8* getelementptr
inbounds ([3 x i8], [3 x i8]* @fmt.10, i32 0, i32 0), i32
%struct_field_value)
  %temp20 = load %struct.node*, %struct.node** %temp
  %getOutNodes = call %struct.List* @get_outNodes(%struct.node*
%temp20)
  store %struct.List* %getOutNodes, %struct.List** %listtemp
  %listtemp21 = load %struct.List*, %struct.List** %listtemp
  %0 = call i32 @l_size(%struct.List* %listtemp21)
  %printf22 = call i32 (i8*, ...) @printf(i8* getelementptr
inbounds ([4 x i8], [4 x i8]* @fmt.8, i32 0, i32 0), i32 %0)
  store i32 0, i32* %i
  br label %while23

while23:                                          ; preds =
%merge, %while_body
  %i58 = load i32, i32* %i
  %listtemp59 = load %struct.List*, %struct.List** %listtemp
  %1 = call i32 @l_size(%struct.List* %listtemp59)
  %tmp60 = icmp slt i32 %i58, %1
  br i1 %tmp60, label %while_body24, label %merge61
```

```
while_body24:                                     ; preds =
%while23
  %listtemp25 = load %struct.List*, %struct.List** %listtemp
  %i26 = load i32, i32* %i
  %val_ptr = call i8* @l_get(%struct.List* %listtemp25, i32
%i26)
  store i8* %val_ptr, i8** %tmpstring
  %g127 = load %struct.graph*, %struct.graph** %g11
  %char_n_val_pointer = load i8*, i8** %tmpstring
  %2 = call %struct.node* @getNode(%struct.graph* %g127, i8*
%char_n_val_pointer)
  store %struct.node* %2, %struct.node** %temp2
  %temp228 = load %struct.node*, %struct.node** %temp2
  %getData29 = call i8* @get_data(%struct.node* %temp228)
  %d_ptr30 = bitcast i8* %getData29 to %nodedata*
  %d_ptr31 = load %nodedata, %nodedata* %d_ptr30
  store %nodedata %d_ptr31, %nodedata* %sourcedata2
  %struct_field_pointer32 = getelementptr inbounds %nodedata,
%nodedata* %sourcedata2, i32 0, i32 1
  %struct_field_value33 = load i32, i32*
%struct_field_pointer32
  %struct_field_pointer34 = getelementptr inbounds %nodedata,
%nodedata* %sourcedata, i32 0, i32 1
  %struct_field_value35 = load i32, i32*
%struct_field_pointer34
  %g136 = load %struct.graph*, %struct.graph** %g11
  %temp37 = load %struct.node*, %struct.node** %temp
  %temp238 = load %struct.node*, %struct.node** %temp2
  %getWeight = call i32 @getWeight(%struct.graph* %g136,
%struct.node* %temp37, %struct.node* %temp238)
  %tmp39 = add i32 %struct_field_value35, %getWeight
  %tmp40 = icmp sgt i32 %struct_field_value33, %tmp39
  br i1 %tmp40, label %then, label %else

merge:                                            ; preds =
%else, %then
  %i56 = load i32, i32* %i
  %tmp57 = add i32 %i56, 1
  store i32 %tmp57, i32* %i
  br label %while23

then:                                             ; preds =
%while_body24
  %struct_field_pointer41 = getelementptr inbounds %nodedata,
%nodedata* %sourcedata, i32 0, i32 1
  %struct_field_value42 = load i32, i32*
%struct_field_pointer41
  %g143 = load %struct.graph*, %struct.graph** %g11
  %temp44 = load %struct.node*, %struct.node** %temp
  %temp245 = load %struct.node*, %struct.node** %temp2
  %getWeight46 = call i32 @getWeight(%struct.graph* %g143,
%struct.node* %temp44, %struct.node* %temp245)
  %tmp47 = add i32 %struct_field_value42, %getWeight46
  %struct_field_pointer48 = getelementptr inbounds %nodedata,
%nodedata* %sourcedata2, i32 0, i32 1
```

```
store i32 %tmp47, i32* %struct_field_pointer48
  %temp249 = load %struct.node*, %struct.node** %temp2
  %sourcedata250 = load %nodedata, %nodedata* %sourcedata2
  %malloccall.51 = tail call i8* @malloc(i32 ptrtoint
(%nodedata* getelementptr (%nodedata, %nodedata* null, i32 1)
to i32))
  %tmp52 = bitcast i8* %malloccall.51 to %nodedata*
  store %nodedata %sourcedata250, %nodedata* %tmp52
  %ptr53 = bitcast %nodedata* %tmp52 to i8*
  call void @set_data(%struct.node* %temp249, i8* %ptr53)
  %p54 = load %struct.pqueue*, %struct.pqueue** %p
  %temp255 = load %struct.node*, %struct.node** %temp2
  call void @pq_push(%struct.pqueue* %p54, %struct.node*
%temp255)
  br label %merge

else:                                             ; preds =
%while_body24
  br label %merge

merge61:                                          ; preds =
%while23
  br label %while

merge64:                                          ; preds =
%while
  ret void
}

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

6.8.3 Dijkstra's Algorithm Output

```
Vertex              Distance from source
a                       0
c                       2
b                       1
d                       7
```

6.9 Breadth-First Search Algorithm: Source Language

BFS.yg is stored in the **example code** directory. Shown below is only the BFS algorithm. In the file, the graph to be analyzed is built in the main method.

```
void BFS(graph<int> g1, node<int> source) {
        Queue<node<int>> p;
        list<string> listtemp;
        node<int> temp;
        node<int> temp2;
        int i;
        string tmpstring;

        p = new Queue<node<int>>();
        /* set distance of source vertex to 0 */

        source.modifyVisited(true);
        p.qadd(source);

        prints("BFS traversal of the graph: ");
        while(p.qsize() != 0) {
                temp = p.qfront();
                printstring(temp@name);
                p.qremove();
                listtemp = temp@outNodes;
                for (i = 0; i < listtemp.lsize(); i = i + 1) {
                        tmpstring = listtemp.l_get(i);
                        temp2 = g1~_tmpstring;
                        if(temp2@visited == false) {
                                temp2.modifyVisited(true);
                                p.qadd(temp2);
                        }
                }
                if (p.qsize() != 0) {
                        printstring (" -> ");
                }
        }
}
```

6.9.1 Breadth-First Search Algorithm: LLVM

```
; ModuleID = 'MicroC'

%struct.QueueId = type { %struct.Node*, %struct.Node*, i32 }
%struct.Node = type { i8*, %struct.Node* }
%struct.List = type { %struct.ListNode*, i32 }
%struct.ListNode = type { i8*, %struct.ListNode* }
%struct.pqueue = type { %struct.node**, i32, i32 }
%struct.node = type { i8*, i8, %struct.map*, %struct.map*, i8*
}
%struct.map = type { i32, i32, i8**, i32* }
%struct.graph = type { %struct.List.5*, i32 }
%struct.List.5 = type { %struct.ListNode.4*, i32 }
%struct.ListNode.4 = type { i8*, %struct.ListNode.4* }

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.2 = private unnamed_addr constant [3 x i8] c"%d\00"
@fmt.3 = private unnamed_addr constant [3 x i8] c"%s\00"
@fmt.4 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@str = private unnamed_addr constant [2 x i8] c"a\00"
@str.5 = private unnamed_addr constant [2 x i8] c"b\00"
@str.6 = private unnamed_addr constant [2 x i8] c"c\00"
@str.7 = private unnamed_addr constant [2 x i8] c"d\00"
@fmt.8 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.9 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.10 = private unnamed_addr constant [3 x i8] c"%d\00"
@fmt.11 = private unnamed_addr constant [3 x i8] c"%s\00"
@fmt.12 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@str.13 = private unnamed_addr constant [29 x i8] c"BFS
traversal of the graph: \00"
@str.14 = private unnamed_addr constant [5 x i8] c" -> \00"

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

declare %struct.QueueId* @initQueueId()

declare void @enqueue(%struct.QueueId*, i8*)

declare void @dequeue(%struct.QueueId*)

declare i8* @front(%struct.QueueId*)

declare i32 @q_size(%struct.QueueId*)

declare %struct.List* @l_init()

declare void @l_add(%struct.List*, i8*)

declare void @l_delete(%struct.List*, i32)

declare i8* @l_get(%struct.List*, i32)
```

```
declare i32 @l_size(%struct.List*)

declare void @print_list(%struct.List*)

declare %struct.pqueue* @pq_init()

declare void @pq_push(%struct.pqueue*, %struct.node*)

declare %struct.node* @pq_delete(%struct.pqueue*)

declare i32 @p_size(%struct.pqueue*)

declare %struct.node* @n_init(i8*)

declare void @set_data(%struct.node*, i8*)

declare i8* @get_name(%struct.node*)

declare i1 @get_visited(%struct.node*)

declare void @modify_visited(%struct.node*, i1)

declare i8* @get_data(%struct.node*)

declare %struct.List* @get_inNodes(%struct.node*)

declare %struct.List* @get_outNodes(%struct.node*)

declare %struct.graph* @g_init()

declare void @addNode(%struct.graph*, %struct.node*)

declare void @removeNode(%struct.graph*, %struct.node*)

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

declare void @removeEdge(%struct.graph*, %struct.node*,
%struct.node*)

declare void @printGraph(%struct.graph*)

declare i1 @isEmpty(%struct.graph*)

declare i32 @size(%struct.graph*)

declare i1 @contains(%struct.graph*, i8*)

declare %struct.node* @getNode(%struct.graph*, i8*)

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

```
define i32 @main() {
entry:
  %g1 = alloca %struct.graph*
  %a1 = alloca i8*
  %a2 = alloca i8*
  %a3 = alloca i8*
  %a4 = alloca i8*
  %i = alloca i32
  %a = alloca %struct.node*
  %b = alloca %struct.node*
  %c = alloca %struct.node*
  %d = alloca %struct.node*
  store i8* getelementptr inbounds ([2 x i8], [2 x i8]* @str,
i32 0, i32 0), i8** %a1
  store i8* getelementptr inbounds ([2 x i8], [2 x i8]* @str.5,
i32 0, i32 0), i8** %a2
  store i8* getelementptr inbounds ([2 x i8], [2 x i8]* @str.6,
i32 0, i32 0), i8** %a3
  store i8* getelementptr inbounds ([2 x i8], [2 x i8]* @str.7,
i32 0, i32 0), i8** %a4
  %init = call %struct.graph* @g_init()
  store %struct.graph* %init, %struct.graph** %g1
  %char_n_val_pointer = load i8*, i8** %a1
  %init1 = call %struct.node* @n_init(i8* %char_n_val_pointer)
  %node_alloca = alloca %struct.node*
  store %struct.node* %init1, %struct.node** %node_alloca
  %graph_pointer = load %struct.graph*, %struct.graph** %g1
  %node_pointer = load %struct.node*, %struct.node**
%node_alloca
  call void @addNode(%struct.graph* %graph_pointer,
%struct.node* %node_pointer)
  %char_n_val_pointer2 = load i8*, i8** %a2
  %init3 = call %struct.node* @n_init(i8* %char_n_val_pointer2)
  %node_alloca4 = alloca %struct.node*
  store %struct.node* %init3, %struct.node** %node_alloca4
  %graph_pointer5 = load %struct.graph*, %struct.graph** %g1
  %node_pointer6 = load %struct.node*, %struct.node**
%node_alloca4
  call void @addNode(%struct.graph* %graph_pointer5,
%struct.node* %node_pointer6)
  %char_n_val_pointer7 = load i8*, i8** %a3
  %init8 = call %struct.node* @n_init(i8* %char_n_val_pointer7)
  %node_alloca9 = alloca %struct.node*
  store %struct.node* %init8, %struct.node** %node_alloca9
  %graph_pointer10 = load %struct.graph*, %struct.graph** %g1
  %node_pointer11 = load %struct.node*, %struct.node**
%node_alloca9
  call void @addNode(%struct.graph* %graph_pointer10,
%struct.node* %node_pointer11)
  %char_n_val_pointer12 = load i8*, i8** %a4
  %init13 = call %struct.node* @n_init(i8*
%char_n_val_pointer12)
  %node_alloca14 = alloca %struct.node*
  store %struct.node* %init13, %struct.node** %node_alloca14
  %graph_pointer15 = load %struct.graph*, %struct.graph** %g1
```

```
%node_pointer16 = load %struct.node*, %struct.node**
%node_alloca14
  call void @addNode(%struct.graph* %graph_pointer15,
%struct.node* %node_pointer16)
  %g117 = load %struct.graph*, %struct.graph** %g1
  %char_n_val_pointer18 = load i8*, i8** %a1
  %0 = call %struct.node* @getNode(%struct.graph* %g117, i8*
%char_n_val_pointer18)
  store %struct.node* %0, %struct.node** %a
  %g119 = load %struct.graph*, %struct.graph** %g1
  %char_n_val_pointer20 = load i8*, i8** %a2
  %1 = call %struct.node* @getNode(%struct.graph* %g119, i8*
%char_n_val_pointer20)
  store %struct.node* %1, %struct.node** %b
  %g121 = load %struct.graph*, %struct.graph** %g1
  %char_n_val_pointer22 = load i8*, i8** %a3
  %2 = call %struct.node* @getNode(%struct.graph* %g121, i8*
%char_n_val_pointer22)
  store %struct.node* %2, %struct.node** %c
  %g123 = load %struct.graph*, %struct.graph** %g1
  %char_n_val_pointer24 = load i8*, i8** %a4
  %3 = call %struct.node* @getNode(%struct.graph* %g123, i8*
%char_n_val_pointer24)
  store %struct.node* %3, %struct.node** %d
  %g125 = load %struct.graph*, %struct.graph** %g1
  %e1_val_pointer = load %struct.node*, %struct.node** %a
  %e2_val_pointer = load %struct.node*, %struct.node** %b
  call void @addEdge(%struct.graph* %g125, %struct.node*
%e1_val_pointer, %struct.node* %e2_val_pointer, i32 1)
  %g126 = load %struct.graph*, %struct.graph** %g1
  %e1_val_pointer27 = load %struct.node*, %struct.node** %a
  %e2_val_pointer28 = load %struct.node*, %struct.node** %c
  call void @addEdge(%struct.graph* %g126, %struct.node*
%e1_val_pointer27, %struct.node* %e2_val_pointer28, i32 2)
  %g129 = load %struct.graph*, %struct.graph** %g1
  %e1_val_pointer30 = load %struct.node*, %struct.node** %b
  %e2_val_pointer31 = load %struct.node*, %struct.node** %a
  call void @addEdge(%struct.graph* %g129, %struct.node*
%e1_val_pointer30, %struct.node* %e2_val_pointer31, i32 1)
  %g132 = load %struct.graph*, %struct.graph** %g1
  %e1_val_pointer33 = load %struct.node*, %struct.node** %b
  %e2_val_pointer34 = load %struct.node*, %struct.node** %d
  call void @addEdge(%struct.graph* %g132, %struct.node*
%e1_val_pointer33, %struct.node* %e2_val_pointer34, i32 10)
  %g135 = load %struct.graph*, %struct.graph** %g1
  %e1_val_pointer36 = load %struct.node*, %struct.node** %c
  %e2_val_pointer37 = load %struct.node*, %struct.node** %a
  call void @addEdge(%struct.graph* %g135, %struct.node*
%e1_val_pointer36, %struct.node* %e2_val_pointer37, i32 2)
  %g138 = load %struct.graph*, %struct.graph** %g1
  %e1_val_pointer39 = load %struct.node*, %struct.node** %c
  %e2_val_pointer40 = load %struct.node*, %struct.node** %d
```

```
call void @addEdge(%struct.graph* %g138, %struct.node*
%e1_val_pointer39, %struct.node* %e2_val_pointer40, i32 5)
  %g141 = load %struct.graph*, %struct.graph** %g1
  %e1_val_pointer42 = load %struct.node*, %struct.node** %d
  %e2_val_pointer43 = load %struct.node*, %struct.node** %b
  call void @addEdge(%struct.graph* %g141, %struct.node*
%e1_val_pointer42, %struct.node* %e2_val_pointer43, i32 10)
  %g144 = load %struct.graph*, %struct.graph** %g1
  %e1_val_pointer45 = load %struct.node*, %struct.node** %d
  %e2_val_pointer46 = load %struct.node*, %struct.node** %c
  call void @addEdge(%struct.graph* %g144, %struct.node*
%e1_val_pointer45, %struct.node* %e2_val_pointer46, i32 5)
  %a47 = load %struct.node*, %struct.node** %a
  %g148 = load %struct.graph*, %struct.graph** %g1
  call void @BFS(%struct.graph* %g148, %struct.node* %a47)
  ret i32 0
}

define void @BFS(%struct.graph* %g1, %struct.node* %source) {
entry:
  %g11 = alloca %struct.graph*
  store %struct.graph* %g1, %struct.graph** %g11
  %source2 = alloca %struct.node*
  store %struct.node* %source, %struct.node** %source2
  %p = alloca %struct.QueueId*
  %listtemp = alloca %struct.List*
  %temp = alloca %struct.node*
  %temp2 = alloca %struct.node*
  %i = alloca i32
  %tmpstring = alloca i8*
  %init = call %struct.QueueId* @initQueueId()
  store %struct.QueueId* %init, %struct.QueueId** %p
  %source3 = load %struct.node*, %struct.node** %source2
  call void @modify_visited(%struct.node* %source3, i1 true)
  %p4 = load %struct.QueueId*, %struct.QueueId** %p
  %source5 = load %struct.node*, %struct.node** %source2
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1**
getelementptr (i1*, i1** null, i32 1) to i32))
  %tmp = bitcast i8* %malloccall to %struct.node**
  store %struct.node* %source5, %struct.node** %tmp
  %ptr = bitcast %struct.node** %tmp to i8*
  call void @enqueue(%struct.QueueId* %p4, i8* %ptr)
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr
inbounds ([4 x i8], [4 x i8]* @fmt.9, i32 0, i32 0), i8*
getelementptr inbounds ([29 x i8], [29 x i8]* @str.13, i32 0,
i32 0))
  br label %while

while:                                            ; preds =
%merge34, %entry
  %p38 = load %struct.QueueId*, %struct.QueueId** %p
  %0 = call i32 @q_size(%struct.QueueId* %p38)
  %tmp39 = icmp ne i32 %0, 0
  br i1 %tmp39, label %while_body, label %merge40
```

```
while_body:                                       ; preds =
%while
  %p6 = load %struct.QueueId*, %struct.QueueId** %p
  %val_ptr = call i8* @front(%struct.QueueId* %p6)
  %d_ptr = bitcast i8* %val_ptr to %struct.node**
  %d_ptr7 = load %struct.node*, %struct.node** %d_ptr
  store %struct.node* %d_ptr7, %struct.node** %temp
  %temp8 = load %struct.node*, %struct.node** %temp
  %getName = call i8* @get_name(%struct.node* %temp8)
  %printf9 = call i32 (i8*, ...) @printf(i8* getelementptr
inbounds ([3 x i8], [3 x i8]* @fmt.11, i32 0, i32 0), i8*
%getName)
  %p10 = load %struct.QueueId*, %struct.QueueId** %p
  call void @dequeue(%struct.QueueId* %p10)
  %temp11 = load %struct.node*, %struct.node** %temp
  %getOutNodes = call %struct.List* @get_outNodes(%struct.node*
%temp11)
  store %struct.List* %getOutNodes, %struct.List** %listtemp
  store i32 0, i32* %i
  br label %while12

while12:                                          ; preds =
%merge, %while_body
  %i28 = load i32, i32* %i
  %listtemp29 = load %struct.List*, %struct.List** %listtemp
  %1 = call i32 @l_size(%struct.List* %listtemp29)
  %tmp30 = icmp slt i32 %i28, %1
  br i1 %tmp30, label %while_body13, label %merge31

while_body13:                                     ; preds =
%while12
  %listtemp14 = load %struct.List*, %struct.List** %listtemp
  %i15 = load i32, i32* %i
  %val_ptr16 = call i8* @l_get(%struct.List* %listtemp14, i32
%i15)
  store i8* %val_ptr16, i8** %tmpstring
  %g117 = load %struct.graph*, %struct.graph** %g11
  %char_n_val_pointer = load i8*, i8** %tmpstring
  %2 = call %struct.node* @getNode(%struct.graph* %g117, i8*
%char_n_val_pointer)
  store %struct.node* %2, %struct.node** %temp2
  %temp218 = load %struct.node*, %struct.node** %temp2
  %getVisited = call i1 @get_visited(%struct.node* %temp218)
  %tmp19 = icmp eq i1 %getVisited, false
  br i1 %tmp19, label %then, label %else

merge:                                            ; preds =
%else, %then
  %i26 = load i32, i32* %i
  %tmp27 = add i32 %i26, 1
  store i32 %tmp27, i32* %i
  br label %while12

then:                                             ; preds =
%while_body13
```

```
%temp220 = load %struct.node*, %struct.node** %temp2
  call void @modify_visited(%struct.node* %temp220, i1 true)
  %p21 = load %struct.QueueId*, %struct.QueueId** %p
  %temp222 = load %struct.node*, %struct.node** %temp2
  %malloccall.23 = tail call i8* @malloc(i32 ptrtoint (i1**
getelementptr (i1*, i1** null, i32 1) to i32))
  %tmp24 = bitcast i8* %malloccall.23 to %struct.node**
  store %struct.node* %temp222, %struct.node** %tmp24
  %ptr25 = bitcast %struct.node** %tmp24 to i8*
  call void @enqueue(%struct.QueueId* %p21, i8* %ptr25)
  br label %merge

else:                                             ; preds =
%while_body13
  br label %merge

merge31:                                          ; preds =
%while12
  %p32 = load %struct.QueueId*, %struct.QueueId** %p
  %3 = call i32 @q_size(%struct.QueueId* %p32)
  %tmp33 = icmp ne i32 %3, 0
  br i1 %tmp33, label %then35, label %else37

merge34:                                          ; preds =
%else37, %then35
  br label %while

then35:                                           ; preds =
%merge31
  %printf36 = call i32 (i8*, ...) @printf(i8* getelementptr
inbounds ([3 x i8], [3 x i8]* @fmt.11, i32 0, i32 0), i8*
getelementptr inbounds ([5 x i8], [5 x i8]* @str.14, i32 0, i32
0))
  br label %merge34

else37:                                           ; preds =
%merge31
  br label %merge34

merge40:                                          ; preds =
%while
  ret void
}
```

6.9.2 Breadth First Search Output

```
BFS traversal of the graph:
a -> b -> c -> d
```

7 Lessons Learned

7.1 Nancy Xu

- I believe that my coding and debugging knowledge improved a lot in doing my project. I learned techniques on how to debug not only my OCaml code, but also how to debug errors or segmentation defaults in my language.
- Understand codegen and semant fully before you begin to implement it
- Compile a C program that performs the function you want to LLVM first, analyze the functions utilized, and then follow that structure as you implement your language.
- Debugging is a very slow process. Be patient, and print things out to screen as much as possible so you can see what goes on behind the scenes.

7.2 Wanlin Xie

- Test your code on VirtualBox if running the code is not working on your local environment and there is no obvious solution. Sometimes the issue is an environmental problem you can spend days to fix with no progress.
- Possess a good understanding of the other files in your project directory (other than the parser, scanner, etc.) such as the testall.sh and microc.sh and how they work to link your files together.
- Generating the clang output of a test program to see how to implement your language in codegen is very useful so you can figure out what Llvm functions you will need to use.

7.3 Yiming Sun

- Understand the MicroC compiler thoroughly before attempting to work on the compiler back-end of your language.
- Utilize a vertical development process (instead of a horizontal one) so that features can be built up layer by layer – and problems can be solved in manageable chunks.
- Start early – especially if it's your first time coding in a functional language, everything will take three times as long as what you planned it out to be.

8 Appendix

Makefile

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

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

all : yeezygraph.native printbig.o

yeezygraph.native :
	ocamlbuild -use-ocamlfind -pkgs
llvm,llvm.analysis,llvm.linker,llvm.bitreader,llvm.irreader -cflags -w,+a-4
\
		yeezygraph.native

# "make clean" removes all generated files

.PHONY : clean
clean :
	ocamlbuild -clean
	rm -rf testall.log *.diff yeezygraph scanner.ml parser.ml parser.mli
	rm -rf printbig
	rm -rf *.cmx *.cmi *.cmo *.cmx *.o *.s *.ll *.out *.exe


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

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

yeezygraph : $(OBJS)
	ocamlfind ocamlopt -linkpkg -package llvm.linker -package
llvm.bitreader -package llvm.irreader -package llvm -package llvm.analysis
$(OBJS) -o yeezygraph

scanner.ml : scanner.mll
	ocamllex scanner.mll

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

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

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

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

```
# Testing the "printbig" example

printbig : printbig.c
	cc -o printbig -DBUILD_TEST printbig.c

### Generated by "ocamldep *.ml *.mli" after building scanner.ml and
parser.ml
ast.cmo :
ast.cmx :
codegen.cmo : ast.cmo node.bc graph.bc linkedlist.bc map.bc queue.bc
pqueue.bc
codegen.cmx : ast.cmx node.bc graph.bc linkedlist.bc map.bc queue.bc
pqueue.bc
yeezygraph.cmo : semant.cmo scanner.cmo parser.cmi codegen.cmo ast.cmo
yeezygraph.cmx : semant.cmx scanner.cmx parser.cmx codegen.cmx ast.cmx
parser.cmo : ast.cmo parser.cmi
parser.cmx : ast.cmx parser.cmi
scanner.cmo : parser.cmi
scanner.cmx : parser.cmx
semant.cmo : ast.cmo
semant.cmx : ast.cmx
parser.cmi : ast.cmo

# Building the tarball

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

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

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

TARFILES = ast.ml codegen.ml Makefile yeezygraph.ml parser.mly README
scanner.mll \
	semant.ml testall.sh $(TESTFILES:%=tests/%) printbig.c arcade-
font.pbm \
	font2c

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

testall.sh

```
#!/bin/sh

# Regression testing script for MicroC
# 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@3.7/bin/lli-3.7"

# Path to the LLVM compiler
# LLC="llc"
LLC="/usr/local/opt/llvm@3.7/bin/llc-3.7"

# Path to the C compiler
CC="clang"

# Path to the microc compiler.  Usually "./microc.native"
# Try "_build/microc.native" if ocamlbuild was unable to create a symbolic
link.
YEEZYGRAPH="./yeezygraph.native"
#MICROC="_build/microc.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] [.yg 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/.yg//'`
    reffile=`echo $1 | sed 's/.yg$//'`
    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 ${basename}.out" &&
    Run "$YEEZYGRAPH" "<" $1 ">" "${basename}.ll" &&
    Run "$LLC" "${basename}.ll" ">" "${basename}.s" &&
    Run "$CC" "-o" "${basename}.exe" "${basename}.s" "queue.bc" "pqueue.bc"
"linkedlist.bc" "graph.bc" "node.bc" "map.bc"  &&
```

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

    echo -n "$basename..."

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

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&
    RunFail "$YEEZYGRAPH" "<" $1 "2>" "${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 printbig.o ]
then
    echo "Could not find printbig.o"
    echo "Try \"make printbig.o\""
    exit 1
fi

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

yeezygraph.sh

```
#!/bin/bash

clang -emit-llvm -o linkedlist.bc -c c/linkedlist.c
clang -emit-llvm -o node.bc -c c/node.c
clang -emit-llvm -o graph.bc -c c/graph.c
clang -emit-llvm -o queue.bc -c c/queue.c
clang -emit-llvm -o map.bc -c c/map.c
clang -emit-llvm -o pqueue.bc -c c/pqueue.c

./yeezygraph.native <$1> a.ll
llvm-link linkedlist.bc node.bc graph.bc queue.bc map.bc pqueue.bc a.ll -S
> run.ll
./a.out
rm a.ll
rm run.ll
rm ./a.out
```

yeezygraph.ml

```
(* Top-level of the MicroC compiler: scan & parse the input,
   check the resulting AST, generate LLVM IR, and dump the module *)

type action = Ast | LLVM_IR | Compile

let _ =
  let action = if Array.length Sys.argv > 1 then
    List.assoc Sys.argv.(1) [ ("-a", Ast);    (* Print the AST only *)
                              ("-l", LLVM_IR);  (* Generate LLVM, don't check
*)
                              ("-c", Compile) ] (* Generate, check LLVM IR *)
  else Compile in
  let lexbuf = Lexing.from_channel stdin in
  let ast = Parser.program Scanner.token lexbuf in
  Semant.check ast;
  match action with
    Ast -> print_string (Ast.string_of_program ast)
  | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate
ast))
  | Compile -> let m = Codegen.translate ast in
    Llvm_analysis.assert_valid_module m;
    print_string (Llvm.string_of_llmodule m)
```

scanner.mll

```
(* Ocamllex scanner for MicroC *)

{ open Parser }
```

```
let digit = ['0'-'9']

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"     { comment lexbuf }           (* Comments *)

| '('      { LPAREN }           | ')'      { RPAREN }
| '['      { LBRACKET }         | ']'      { RBRACKET }
| '{'      { LBRACE }           | '}'      { RBRACE }
| ';'      { SEMI }             | ','      { COMMA }

| '+'      { PLUS }             | '-'      { MINUS }
| '*'      { TIMES }            | '/'      { DIVIDE }
| '='      { ASSIGN }

| "=="     { EQ }               | "!="     { NEQ }
| '<'      { LT }               | "<="     { LEQ }
| ">"      { GT }               | ">="     { GEQ }
| "&&"     { AND }              | "||"     { OR }
| "!"      { NOT }

| "if"     { IF }               | "else"   { ELSE }
| "for"    { FOR }              | "while"  { WHILE }
| "return" { RETURN }           | "new"    { NEW }
| '.'     { DOT }

| "int"    { INT }
| "bool"   { BOOL }             | "true"   { TRUE }             | "false"
{ FALSE }
| "float"  { FLOAT }
| "string" { STRING }

| "struct" { STRUCT }           | "~"     { TILDE }
| "Queue"  { QUEUE }            | "pqueue" { PQUEUE }
| "graph" { GRAPH }             | "node"   { NODE }
| "list"   { LIST }


| "~+" { ADD_NODE }             | "~-" { REMOVE_NODE }
| "->" { ADD_EDGE }             | "!->" { REMOVE_EDGE }

| "~_" { UNDERSCORE }
| "@name" { GET_NAME }      | "@visited" { GET_VISITED }
| "@data" { GET_DATA }      | "@inNodes" { GET_INNODES }
| "@outNodes" { GET_OUTNODES }

| "INFINITY" {INF }         | "NEG-INFINITY" {NEG_INF}



| "new" { NEW }
| "void"   { VOID }
```

```
| ['0'-'9']+ as lxm { INT_LITERAL(int_of_string lxm) }
| digit+('.')digit+ as lxm { FLOAT_LITERAL(float_of_string lxm) }
| '"'([^'"']* as lxm)'"' { STR_LITERAL(lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

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

parser.mly

```
/* Ocamlyacc parser for MicroC */

%{
  open Ast

  let get_1_3 (a,_,_) = a
  let get_2_3 (_,a,_) = a
  let get_3_3 (_,_,a) = a
%}


/* Punctuation tokens */
%token SEMI LPAREN RPAREN LBRACKET RBRACKET LBRACE RBRACE COMMA

/* Primitive datatype tokens */
%token INT BOOL FLOAT STRING

/* Struct datatype tokens */
%token STRUCT TILDE

/* Graph tokens */
%token ADD_NODE REMOVE_NODE ADD_EDGE REMOVE_EDGE GRAPH NODE

/* Queue datatype tokens */
%token QUEUE

/* List datatype tokens */
%token LIST

/* PQueue datatype tokens */
%token PQUEUE

/* Arithmetic tokens */
%token PLUS MINUS TIMES DIVIDE ASSIGN NOT

/* Logical tokens */
%token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR
```

```
/* Control flow tokens */
%token IF ELSE FOR WHILE

/* Function tokens */
%token RETURN VOID NEW DOT

%token INF NEG_INF



%token UNDERSCORE
%token GET_NAME GET_VISITED GET_DATA GET_INNODES GET_OUTNODES

%token <int> INT_LITERAL
%token <float> FLOAT_LITERAL
%token <string> STR_LITERAL
%token <string> ID
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE
%right NOT NEG
%left DOT
%left TILDE
%nonassoc GET_NAME GET_VISITED GET_DATA GET_INNODES GET_OUTNODES LBRACKET
RBRACKET
%left ADD_EDGE REMOVE_EDGE
%nonassoc UNDERSCORE
%left ADD_NODE REMOVE_NODE


%start program
%type <Ast.program> program

%%

program:
  decls EOF { $1 }

decls:
   /* nothing */ { [], [], [] }
 | decls vdecl { ($2 :: get_1_3($1)), get_2_3($1), get_3_3($1) }
 | decls fdecl { get_1_3($1), ($2 :: get_2_3($1)), get_3_3($1) }
 | decls sdecl { get_1_3($1), get_2_3($1), $2 :: get_3_3($1) }
```

```
fdecl:
   typ ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list RBRACE
     { { typ = $1;
              fname = $2;
              formals = $4;
              locals = List.rev $7;
              body = List.rev $8 } }

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

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


typ:
    INT { Int }
  | FLOAT { Float }
  | BOOL { Bool }
  | STRING { String }
  | VOID { Void }
  | QUEUE LT typ GT { QueueType($3)}
  | PQUEUE { PQueueType }
  | STRUCT ID { StructType ($2) }
  | GRAPH LT typ GT { GraphType($3)}
  | NODE LT typ GT { NodeType($3) }
  | LIST LT typ GT { ListType($3) }


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

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


sdecl:
   STRUCT ID LBRACE vdecl_list RBRACE
      { {
        sname = $2;
        sformals = $4; } }


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

```
stmt:
    expr SEMI { Expr $1 }
  | RETURN SEMI { Return Noexpr }
  | RETURN expr SEMI { Return $2 }
  | LBRACE stmt_list RBRACE { Block(List.rev $2) }
  | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
  | IF LPAREN expr RPAREN stmt ELSE stmt    { If($3, $5, $7) }
  | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
     { For($3, $5, $7, $9) }
  | WHILE LPAREN expr RPAREN stmt { While($3, $5) }

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


expr:
    INT_LITERAL           { IntLit($1) }
  | FLOAT_LITERAL         { FloatLit($1) }
  | STR_LITERAL           { StringLit($1) }
  | TRUE                  { BoolLit(true) }
  | FALSE                 { BoolLit(false) }
  | NEW QUEUE LT typ GT LPAREN actuals_opt RPAREN { Queue($4, $7) }
  | NEW PQUEUE LPAREN actuals_opt RPAREN { PQueue($4) }
  | NEW LIST LT typ GT LPAREN actuals_opt RPAREN { List($4,$7) }
  | ID                    { Id($1) }
  | expr PLUS   expr      { Binop($1, Add,   $3) }
  | expr MINUS  expr      { Binop($1, Sub,   $3) }
  | expr TIMES  expr      { Binop($1, Mult,  $3) }
  | expr DIVIDE expr      { Binop($1, Div,   $3) }
  | expr EQ     expr      { Binop($1, Equal, $3) }
  | expr NEQ    expr      { Binop($1, Neq,   $3) }
  | expr LT     expr      { Binop($1, Less,  $3) }
  | expr LEQ    expr      { Binop($1, Leq,   $3) }
  | expr GT     expr      { Binop($1, Greater, $3) }
  | expr GEQ    expr      { Binop($1, Geq,   $3) }
  | expr AND    expr      { Binop($1, And,   $3) }
  | expr OR     expr      { Binop($1, Or,    $3) }
  | MINUS expr %prec NEG  { Unop(Neg, $2) }
  | NOT expr              { Unop(Not, $2) }
  | expr TILDE  ID        { AccessStructField($1, $3) }
  | expr ASSIGN expr      { Assign($1, $3) }
  | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
  | expr DOT ID LPAREN actuals_opt RPAREN { ObjectCall($1, $3, $5) }
  | LPAREN expr RPAREN    { $2 }
  | expr UNDERSCORE ID            { NodeOp($1, AccessNode, $3) }
  | expr GET_NAME                   { NodeOp2($1, GetName) }
  | expr GET_DATA                   { NodeOp2($1, GetData) }
  | expr GET_VISITED                { NodeOp2($1, GetVisited) }
  | expr GET_INNODES                { NodeOp2($1, GetinNodes) }
  | expr GET_OUTNODES                { NodeOp2($1, GetoutNodes) }
  | ID ADD_NODE ID       { GraphOp($1, AddNode, $3) }
```

```
  | ID REMOVE_NODE ID    { GraphOp($1, RemoveNode, $3) }
  | expr LBRACKET INT_LITERAL RBRACKET ADD_EDGE LPAREN ID COMMA ID RPAREN
{ GraphOpAddEdge($1, $3, AddEdge, $7, $9) }
  | ID REMOVE_EDGE LPAREN ID COMMA ID RPAREN          {
GraphOpRemoveEdge($1, RemoveEdge, $4, $6) }
  | NEW GRAPH LT typ GT LPAREN RPAREN { Graph($4) }
  | NEW NODE LT typ GT LPAREN ID RPAREN {Node($7, $4)}
  | INF {Infinity}
  | NEG_INF {NegInfinity}

actuals_opt:
    /* ng */ { [] }
  | actuals_list  { List.rev $1 }

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

ast.ml

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

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

type gop = AddNode | RemoveNode
type gop2 = AddEdge
type gop3 = RemoveEdge

type nop = AccessNode (*underscore, e.g. g1_n1*)
type nop2 = GetName | GetData | GetVisited | GetinNodes | GetoutNodes


type uop = Neg | Not

type typ = Int | Bool | Float | String | Void
          | StructType of string | GraphType of typ | NodeType of typ
          | QueueType of typ | PQueueType | AnyType |  ListType of typ

type bind = typ * string

type expr =
    IntLit of int
  | BoolLit of bool
  | FloatLit of float
  | StringLit of string
  | Queue of typ * expr list
  | List of typ * expr list
  | PQueue of expr list
  | Id of string
  | Binop of expr * op * expr
```

```
  | Unop of uop * expr
  | AccessStructField of expr * string
  | Assign of expr * expr
  | Call of string * expr list
  | ObjectCall of expr * string * expr list
  | Noexpr
  | NodeOp of expr * nop * string (* g1_n1 / g1_n1@name *)
  | NodeOp2 of expr * nop2
  | GraphOp of string * gop * string
  | GraphOpRemoveEdge of string * gop3 * string * string
  | GraphOpAddEdge of expr * int * gop2 * string * string
  | Graph of typ
  | Node of string * typ
  | Infinity
  | NegInfinity

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

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

type struct_decl = {
  sname : string;
  sformals : bind list;
}

type program = bind list * func_decl list * struct_decl list


(* Pretty-printing functions *)

let string_of_op = function
    Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
```

```
  | Geq -> ">="
  | And -> "&&"
  | Or -> "||"


let string_of_nop = function
  | AccessNode -> "~_"

let string_of_nop2 = function
    GetName -> "@name"
  | GetVisited -> "@visited"
  | GetData -> "@data"
  | GetinNodes -> "@inNodes"
  | GetoutNodes -> "@outNodes"

let string_of_gop = function
    AddNode -> "~+"
  | RemoveNode -> "~-"



let string_of_gop2 = function
    AddEdge -> "->"

let string_of_gop3 = function
    RemoveEdge -> "!->"

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

let rec string_of_typ = function
    Int -> "int"
  | Float -> "float"
  | Bool -> "bool"
  | String -> "string"
  | Void -> "void"
  | GraphType(typ) -> "graph " ^ string_of_typ typ
  | StructType(s) -> s
  | QueueType(typ) -> "queue " ^ string_of_typ typ
  | ListType(typ) -> "list " ^ string_of_typ typ
  | PQueueType -> "pqueue"
  | NodeType(typ) -> "node " ^ string_of_typ typ
  | AnyType -> "AnyType"

let rec string_of_expr = function
    IntLit(l) -> string_of_int l
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | FloatLit(l) -> string_of_float l
  | StringLit(s) -> s
  | Id(s) -> s
```

```
  | Queue(typ, e1) -> "new " ^ "Queue" ^ "<" ^ string_of_typ typ ^ ">" ^
"(" ^ String.concat ", " (List.map string_of_expr e1) ^ ")"
  | PQueue(el) -> "new" ^ "pqueue" ^ "(" ^ String.concat ", " (List.map
string_of_expr el) ^ ")"
  | List(typ, el) -> "new" ^ "list" ^ "<" ^ string_of_typ typ ^ ">" ^ "(" ^
String.concat "," (List.map string_of_expr el) ^ ")"
  | 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
  | AccessStructField(v, e) -> string_of_expr v ^ "~" ^ e
  | Assign(v, e) -> string_of_expr v ^ " = " ^ string_of_expr e
  | Call(f, el) -> f ^ "(" ^ String.concat ", " (List.map string_of_expr
el) ^ ")"
  | ObjectCall(o, f, e1) -> string_of_expr o ^ "." ^ f ^ "(" ^
String.concat ", " (List.map string_of_expr e1) ^ ")"
  | Noexpr -> ""
  | NodeOp(e, o, s) -> string_of_expr e ^ string_of_nop o  ^ s
  | NodeOp2(e, o) -> string_of_expr e ^ string_of_nop2 o
  | Graph(typ) -> "new Graph" ^ "<" ^ string_of_typ typ ^ ">"
  | Node(n, typ) -> "new Node" ^ "<" ^ string_of_typ typ ^ ">" ^ "(" ^ n ^
")"
  | GraphOp(s1, o, s2) ->  s1 ^ string_of_gop o ^  s2
  | GraphOpRemoveEdge(s1, o, s2, s3) ->   s1 ^ string_of_gop3 o ^  "(" ^ s2
^ ", " ^ s3 ^ ")"
  | GraphOpAddEdge(s1, i, o, s2, s3) -> string_of_expr s1 ^ string_of_int i
^ string_of_gop2 o ^ "(" ^ s2 ^ ", " ^ s3 ^ ")"
  | Infinity -> "INFINITY"
  | NegInfinity -> "NEG-INFINITY"



let rec string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
  | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^
string_of_stmt s
  | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
      string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
  | For(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


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

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

```
  String.concat "" (List.map string_of_vdecl fdecl.locals) ^
  String.concat "" (List.map string_of_stmt fdecl.body) ^
  "}\n"

let string_of_sdecl sdecl =
  "struct " ^ sdecl.sname ^ " \n{\n" ^
  String.concat "; " (List.map snd sdecl.sformals) ^
  "}\n"

let string_of_program (vars, funcs, structs) =
  String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
  String.concat "\n" (List.map string_of_sdecl structs) ^ "\n" ^
  String.concat "\n" (List.map string_of_fdecl funcs)
```

codegen.ml

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

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

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

Detailed documentation on the OCaml LLVM library:

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

*)

module L = Llvm
module A = Ast


module StringMap = Map.Make(String)

let translate (globals, functions, structs) =
  let context = L.global_context () in (* global data container *)
  let nodecontext = L.global_context () in
  let graphcontext = L.global_context () in
  let the_module = L.create_module context "MicroC" in(* container *)
  let llctx = L.global_context () in

  let qqctx = L.global_context () in
  let queuem = L.MemoryBuffer.of_file "queue.bc" in
  let listm = L.MemoryBuffer.of_file "linkedlist.bc" in
  let llm = Llvm_bitreader.parse_bitcode llctx listm in
  let qqm = Llvm_bitreader.parse_bitcode qqctx queuem in

  let ppctx = L.global_context () in
  let pqueuem = L.MemoryBuffer.of_file "pqueue.bc" in
  let ppm = Llvm_bitreader.parse_bitcode ppctx pqueuem in
```

```
  let nodem = L.MemoryBuffer.of_file "node.bc" in
  let node_module = Llvm_bitreader.parse_bitcode nodecontext nodem in
  let graphm = L.MemoryBuffer.of_file "graph.bc" in
  let graph_module = Llvm_bitreader.parse_bitcode graphcontext graphm in

  let i32_t  = L.i32_type  context
  and i8_t   = L.i8_type   context (* for printf format string *)
  and float_t  = L.double_type context
  and i1_t   = L.i1_type   context
  and void_t = L.void_type context
  and queueid_t = L.pointer_type (match L.type_by_name qqm "struct.QueueId"
with
    None -> raise (Invalid_argument "Option.get queueid") | Some x -> x)
  and pqueue_t = L.pointer_type (match L.type_by_name ppm "struct.pqueue"
with
    None -> raise (Invalid_argument "Option.get pqueue") | Some x -> x)
  and graph_t = L.pointer_type (match L.type_by_name graph_module
"struct.graph" with
    None -> raise (Invalid_argument "Option.get graph") | Some x -> x )
  and node_t = L.pointer_type (match L.type_by_name node_module
"struct.node" with
    None -> raise (Invalid_argument "Option.get node") | Some x -> x )
  and list_t = L.pointer_type (match L.type_by_name llm "struct.List" with
    None -> raise (Invalid_argument "Option.get struct.List") | Some x ->
x) in


let struct_types =
  let add_struct m sdecl =
    let struct_t = L.named_struct_type context sdecl.A.sname
    in StringMap.add sdecl.A.sname struct_t m in
  List.fold_left add_struct StringMap.empty structs in


  let ltype_of_typ = function (* LLVM type for a given AST type *)
      A.Int -> i32_t
    | A.Float -> float_t
    | A.Bool -> i1_t
    | A.String -> L.pointer_type i8_t
    | A.Void -> void_t
    | A.StructType s -> (try StringMap.find s struct_types with Not_found -
> raise (Failure("struct type " ^ s ^ " is undefined") ))
    | A.GraphType _ -> graph_t
    | A.NodeType _  -> node_t
    | A.QueueType _ -> queueid_t
    | A.PQueueType -> pqueue_t
    | A.ListType _ -> list_t
    | A.AnyType -> L.pointer_type i8_t in
```

```
  (* Build struct body*)
  let build_struct_body sdecl =
    let struct_t = StringMap.find sdecl.A.sname struct_types in
    let element_list = Array.of_list(List.map (fun (t, _) -> ltype_of_typ
t) sdecl.A.sformals) in
    L.struct_set_body struct_t element_list true in
    ignore(List.map build_struct_body structs);


  (* struct_indexing_map(struct_type -> field_indexing_map *)
  (* field_indexing_map(field_name -> index) *)
  let struct_indexing_map =
    let fill_struct_indexing_map m this_struct =
      let field_name_list = List.map snd this_struct.A.sformals in
      let add_one i = i + 1 in
      let fill_field_indexing_map (m, i) field_name = StringMap.add
field_name (add_one i) m, add_one i in
      let field_indexing_map = List.fold_left fill_field_indexing_map
(StringMap.empty, -1) field_name_list in
      StringMap.add this_struct.A.sname (fst field_indexing_map) m
    in
    List.fold_left fill_struct_indexing_map StringMap.empty structs
  in



  (* Declare and initialize each global variable; remember its value in a
map *)
  let global_types =
    let global_type m (t, n) = StringMap.add n t m in
    List.fold_left global_type StringMap.empty globals in

  let global_vars =
    let global_var m (t, n) =
      let init = L.const_null (ltype_of_typ t)
      in StringMap.add n (L.define_global n init the_module) m in
    List.fold_left global_var StringMap.empty globals in



  (* Declare printf(), which the print built-in function will call *)
  let printf_t = L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
(* Declare a function type *)
  let printf_func = L.declare_function "printf" printf_t the_module in

  (* built-in queue functions *)
  let initQueueId_t = L.function_type queueid_t [| |] in
  let initQueueId_f = L.declare_function "initQueueId" initQueueId_t
the_module in
  let enqueue_t = L.function_type void_t [| queueid_t; L.pointer_type
i8_t|] in
  let enqueue_f = L.declare_function "enqueue" enqueue_t the_module in
```

```
  let dequeue_t = L.function_type void_t [| queueid_t |] in
  let dequeue_f = L.declare_function "dequeue" dequeue_t the_module in
  let front_t = L.function_type (L.pointer_type i8_t) [| queueid_t |] in
  let front_f = L.declare_function "front" front_t the_module in
  let sizeQ_t = L.function_type i32_t [| queueid_t |] in
  let sizeQ_f = L.declare_function "q_size" sizeQ_t the_module in

  (* built-in linked list functions *)
  let initList_t = L.function_type list_t [| |] in
  let initList_f = L.declare_function "l_init" initList_t the_module in
  let addList_t = L.function_type void_t [| list_t; (L.pointer_type i8_t)
|] in
  let addList_f = L.declare_function "l_add" addList_t the_module in
  let delList_t = L.function_type void_t [| list_t; i32_t |] in
  let delList_f = L.declare_function "l_delete" delList_t the_module in
  let getList_t = L.function_type (L.pointer_type i8_t) [| list_t; i32_t |]
in
  let getList_f = L.declare_function "l_get" getList_t the_module in
  let sizeList_t = L.function_type i32_t [| list_t|] in
  let sizeList_f = L.declare_function "l_size" sizeList_t the_module in
  let printList_t = L.function_type void_t [| list_t|] in
  let printList_f = L.declare_function "print_list" printList_t the_module
in

  (* Pqueue functions *)
  let initPQueue_t = L.function_type pqueue_t [| |] in
  let initPQueue_f = L.declare_function "pq_init" initPQueue_t the_module
in
  let pushPQ_t = L.function_type void_t [| pqueue_t; node_t |] in
  let pushPQ_f = L.declare_function "pq_push" pushPQ_t the_module in
  let deletePQ_t = L.function_type node_t [| pqueue_t |] in
  let deletePQ_f = L.declare_function "pq_delete" deletePQ_t the_module in
  let sizePQ_t = L.function_type i32_t [| pqueue_t |] in
  let sizePQ_f = L.declare_function "p_size" sizePQ_t the_module in

  (* Node functions *)
  let initNode_t = L.function_type node_t [| L.pointer_type i8_t |] in
  let initNode_f = L.declare_function "n_init" initNode_t the_module in
  let setData_t = L.function_type void_t [| node_t; L.pointer_type i8_t|]
in
  let setData_f = L.declare_function "set_data" setData_t the_module in
  let getName_t = L.function_type (L.pointer_type i8_t) [| node_t |] in
  let getName_f = L.declare_function "get_name" getName_t the_module in
  let getVisited_t = L.function_type i1_t [| node_t |] in
  let getVisited_f = L.declare_function "get_visited" getVisited_t
the_module in
  let modifyVisited_t = L.function_type void_t [| node_t; i1_t|] in
  let modifyVisited_f = L.declare_function "modify_visited" modifyVisited_t
the_module in
  let getData_t = L.function_type (L.pointer_type i8_t) [| node_t |] in
  let getData_f = L.declare_function "get_data" getData_t the_module in
  let getinNodes_t = L.function_type (list_t) [| node_t |] in
```

```
  let getinNodes_f = L.declare_function "get_inNodes" getinNodes_t
the_module in
  let getoutNodes_t = L.function_type (list_t) [| node_t |] in
  let getoutNodes_f = L.declare_function "get_outNodes" getoutNodes_t
the_module in



  (* Graph functions *)
  let initGraph_t = L.function_type graph_t [| |] in
  let initGraph_f = L.declare_function "g_init" initGraph_t the_module in
  let addNode_t = L.function_type void_t [| graph_t; node_t |] in
  let addNode_f = L.declare_function "addNode" addNode_t the_module in
  let removeNode_t = L.function_type void_t [| graph_t; node_t |] in
  let removeNode_f = L.declare_function "removeNode" removeNode_t
the_module in
  let addEdge_t = L.function_type void_t [| graph_t; node_t; node_t; i32_t
|] in
  let addEdge_f = L.declare_function "addEdge" addEdge_t the_module in
  let removeEdge_t = L.function_type void_t [| graph_t; node_t; node_t |]
in
  let removeEdge_f = L.declare_function "removeEdge" removeEdge_t
the_module in
  let printGraph_t = L.function_type void_t [| graph_t|] in
  let printGraph_f = L.declare_function "printGraph" printGraph_t
the_module in
  let isEmpty_t = L.function_type i1_t [| graph_t|] in
  let isEmpty_f = L.declare_function "isEmpty" isEmpty_t the_module in
  let size_t = L.function_type i32_t [| graph_t|] in
  let size_f = L.declare_function "size" size_t the_module in
  let contains_t = L.function_type i1_t [| graph_t; L.pointer_type i8_t|]
in
  let contains_f = L.declare_function "contains" contains_t the_module in
  let getNode_t = L.function_type node_t [| graph_t; L.pointer_type i8_t|]
in
  let getNode_f = L.declare_function "getNode" getNode_t the_module in
  let getWeight_t = L.function_type i32_t [| graph_t; node_t; node_t|] in
  let getWeight_f = L.declare_function "getWeight" getWeight_t the_module
in




  (* Define each user-defined function (arguments and return type);
remember in a map *)
  let function_decls =
    let function_decl m fdecl =
      let name = fdecl.A.fname
      and formal_types = Array.of_list
          (List.map (fun (t,_) -> ltype_of_typ t) fdecl.A.formals)
      in let ftype =
          L.function_type (ltype_of_typ fdecl.A.typ) 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.A.fname 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 in
    let str_format_str = L.build_global_stringptr "%s\n" "fmt" builder in

    let int_format_str2 = L.build_global_stringptr "%d" "fmt" builder in
    let str_format_str2 = L.build_global_stringptr "%s" "fmt" builder in

    let float_format_str = L.build_global_stringptr "%f\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 in

      let 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.A.formals
          (Array.to_list (L.params the_function)) in
          List.fold_left add_local formals fdecl.A.locals in


    let local_types =
      let add_type m (t, n) = StringMap.add n t m in
      let formal_types = List.fold_left add_type StringMap.empty
fdecl.A.formals in
          List.fold_left add_type formal_types fdecl.A.locals in


    (* Return the value for a variable or formal argument *)
    let lookup n = try StringMap.find n local_vars
                   with Not_found -> StringMap.find n global_vars
    in

    let lookup_types n = try StringMap.find n local_types
                         with Not_found -> StringMap.find n global_types in
```

```
    let getQueueType = function
       A.QueueType(typ) -> typ
      | _ -> A.Void in

    let getListType = function
      A.ListType(typ) -> typ
    | _ -> A.String in

    let getNodeType = function
       A.NodeType(typ) -> typ
      | _ -> A.Void in

    let idtostring = function
        A.Id s -> s
      | _ -> "" in


    (* Construct code for an expression; return its value *)
    let rec expr builder = function
            A.IntLit i -> L.const_int i32_t i
      | A.BoolLit b -> L.const_int i1_t (if b then 1 else 0)
      | A.StringLit s -> L.build_global_stringptr s "str" builder
      | A.FloatLit f -> L.const_float float_t f
      | A.Noexpr -> L.const_int i32_t 0
      | A.Id s -> L.build_load (lookup s) s builder
      | A.Queue (typ, act) ->
        let d_ltyp = ltype_of_typ typ in
        let queue_ptr = L.build_call initQueueId_f [| |] "init" builder in
        let add_element elem =
          let d_ptr = match typ with
          | A.QueueType _ -> expr builder elem
          | _ ->
            let element = expr builder elem in
            let d_ptr = L.build_malloc d_ltyp "tmp" builder in
            ignore (L.build_store element d_ptr builder); d_ptr in
          let void_d_ptr = L.build_bitcast d_ptr (L.pointer_type i8_t)
"ptr" builder in
          ignore (L.build_call enqueue_f [| queue_ptr; void_d_ptr |] ""
builder)
        in ignore (List.map add_element act);
        queue_ptr
      | A.List (typ, act) ->
        let d_ltyp = ltype_of_typ typ in
        let listptr = L.build_call initList_f [||] "init" builder in
          let add_elmt elmt =
            let d_ptr = match typ with
            A.ListType _ -> expr builder elmt
          | _ ->
            let d_val = expr builder elmt in
            let d_ptr = L.build_malloc d_ltyp "tmp" builder in
            ignore (L.build_store d_val d_ptr builder);
```

```
            d_ptr in

            let void_d_ptr = L.build_bitcast d_ptr (L.pointer_type i8_t)
"ptr" builder in
            ignore (L.build_call addList_f [| listptr; void_d_ptr |] ""
builder) in
          ignore (List.map add_elmt act);
        listptr
      | A.PQueue (act) ->
        let pqptr = L.build_call initPQueue_f [| |] "init" builder in
          let add_elmt elmt =
            let element = expr builder elmt in
          ignore (L.build_call pushPQ_f [| pqptr; element |] "" builder) in
        ignore (List.map add_elmt act);
        pqptr
      | A.Binop (e1, op, e2) ->
         let e1' = expr builder e1
             and e2' = expr builder e2
         in
         let t1 = L.type_of e1' in
             (match op with
               A.Add     -> if t1 = float_t then L.build_fadd else
L.build_add
             | A.Sub     -> if t1 = float_t then L.build_fsub else
L.build_sub
             | A.Mult    -> if t1 = float_t then L.build_fmul else
L.build_mul
         | A.Div     -> if t1 = float_t then L.build_fdiv else L.build_sdiv
             | A.And     -> L.build_and
             | A.Or      -> L.build_or
             | A.Equal   -> if t1 = float_t then L.build_fcmp L.Fcmp.Oeq
else L.build_icmp L.Icmp.Eq
             | A.Neq     -> if t1 = float_t then L.build_fcmp L.Fcmp.One
else L.build_icmp L.Icmp.Ne
             | A.Less    -> if t1 = float_t then L.build_fcmp L.Fcmp.Olt
else L.build_icmp L.Icmp.Slt
             | A.Leq     -> if t1 = float_t then L.build_fcmp L.Fcmp.Ole
else L.build_icmp L.Icmp.Sle
             | A.Greater -> if t1 = float_t then L.build_fcmp L.Fcmp.Ogt
else L.build_icmp L.Icmp.Sgt
             | A.Geq     -> if t1 = float_t then L.build_fcmp L.Fcmp.Oge
else L.build_icmp L.Icmp.Sge ) e1' e2' "tmp" builder
      | A.Unop(op, e) ->
             let e' = expr builder e in
             (match op with
               A.Neg     -> L.build_neg
         | A.Not     -> L.build_not) e' "tmp" builder

      | A.NodeOp(e, nop, s ) ->
          let e_val = expr builder e in
          let s_val = lookup s in
          (match nop with
```

```
            A.AccessNode ->
           let char_s_val_pointer = L.build_load s_val "char_n_val_pointer"
builder in
           let node_pointer = L.build_call getNode_f [| e_val;
char_s_val_pointer|] "" builder in
           node_pointer
          )

      | A.NodeOp2(e, nop2) ->
            let e_val = expr builder e in

            let n_type = getNodeType(lookup_types(idtostring e)) in
            (match nop2 with
              A.GetName ->
                let c_pointer = L.build_call getName_f [| e_val |]
"getName" builder in c_pointer
            | A.GetVisited ->
                let b_pointer = L.build_call getVisited_f [| e_val |]
"getVisited" builder in b_pointer
            | A.GetData ->
                let v_pointer = L.build_call getData_f [| e_val |]
"getData" builder in
                let l_dtyp = ltype_of_typ n_type in
                let d_ptr = L.build_bitcast v_pointer (L.pointer_type
l_dtyp) "d_ptr" builder in
                (L.build_load d_ptr "d_ptr" builder)
            | A.GetinNodes ->
                let v_pointer = L.build_call getinNodes_f [| e_val |]
"getinNodes" builder in v_pointer

            | A.GetoutNodes ->
                let v_pointer = L.build_call getoutNodes_f [| e_val |]
"getOutNodes" builder in
                v_pointer)

      | A.Node(n, _) ->
          let n_val = lookup n in
          let char_n_val_pointer = L.build_load n_val "char_n_val_pointer"
builder in
          let nodeptr = L.build_call initNode_f [| char_n_val_pointer |]
"init_n" builder in nodeptr

      | A.Graph(_) ->
          let graphptr = L.build_call initGraph_f [| |] "init" builder in
graphptr

      | A.GraphOp(g, gop, n) ->
          let g_val = lookup g in
          let n_val = lookup n in
          let char_n_val_pointer = L.build_load n_val "char_n_val_pointer"
builder in
```

```
          (match gop with
           A.AddNode ->

           let n_ptr = L.build_call initNode_f [| char_n_val_pointer |]
"init" builder in
           let n_val_ptr = L.build_alloca (L.type_of n_ptr) "node_alloca"
builder in
           ignore (L.build_store n_ptr n_val_ptr builder);
           let graph_pointer = L.build_load g_val "graph_pointer" builder
in
           let node_pointer = L.build_load n_val_ptr "node_pointer" builder
in
           ignore(L.build_call addNode_f [| graph_pointer; node_pointer |]
"" builder); g_val
          | A.RemoveNode ->
           let n_ptr = L.build_call initNode_f [| char_n_val_pointer |]
"init" builder in
           let n_val_ptr = L.build_alloca (L.type_of n_ptr) "node_alloca"
builder in
           ignore (L.build_store n_ptr n_val_ptr builder);
           let graph_pointer = L.build_load g_val "graph_pointer" builder
in
           let node_pointer = L.build_load n_val_ptr "node_pointer" builder
in
           ignore(L.build_call removeNode_f [| graph_pointer; node_pointer
|] "" builder); g_val

         )
      | A.GraphOpAddEdge(g, i, gop2, e1, e2) ->
        (match gop2 with
          A.AddEdge ->
          let g_val = expr builder g in
          let i_val = L.const_int i32_t i in
          let e1_val = lookup e1 in
          let e1_val_pointer = L.build_load e1_val "e1_val_pointer" builder
in
          let e2_val = lookup e2 in
          let e2_val_pointer = L.build_load e2_val "e2_val_pointer" builder
in
          let g_pointer = L.build_call addEdge_f [| g_val; e1_val_pointer;
e2_val_pointer; i_val|] "" builder in g_pointer
        )

      | A.GraphOpRemoveEdge(g, gop3, e1, e2) ->
        (match gop3 with
          A.RemoveEdge ->
          let g_val = lookup g in
          let g_val_pointer = L.build_load g_val "g_val_pointer" builder in
          let e1_val = lookup e1 in
          let e1_val_pointer = L.build_load e1_val "e1_val_pointer" builder
in
          let e2_val = lookup e2 in
```

```
          let e2_val_pointer = L.build_load e2_val "e2_val_pointer" builder
in
          let g_pointer = L.build_call removeEdge_f [| g_val_pointer;
e1_val_pointer; e2_val_pointer|] "" builder in g_pointer
        )

      | A.AccessStructField(e, field_name) ->
        (match e with
           A.Id s -> let etype = fst(List.find (fun local -> snd(local) =
s) fdecl.A.locals) in
              (match etype with
                  A.StructType struct_type ->
                    let field_indexing_map = StringMap.find struct_type
struct_indexing_map in
                    let index = StringMap.find field_name
field_indexing_map in
                    let struct_llvalue = lookup s in
                    (* %struct_field_pointer = getelementptr
%struct_llvalue, 0, index_number*)
                    let struct_field_pointer = L.build_struct_gep
struct_llvalue index "struct_field_pointer" builder in
                    (* %struct_field_value = load %struct_field_pointer*)
                    let struct_field_value = L.build_load
struct_field_pointer "struct_field_value" builder in
                    struct_field_value
                | _ -> raise (Failure("AccessStructField failed: " ^ s ^
"is not a struct type")))
         | _ -> raise (Failure("AccessStructField failed")))


      | A.Assign (e1, e2) -> let e2' = expr builder e2 in
        (match e1 with
          A.Id s -> ignore (L.build_store e2' (lookup s) builder); e2'
        | A.AccessStructField(e, field_name) ->
          (match e with
            A.Id s -> let etype = fst(List.find (fun local -> snd(local) =
s) fdecl.A.locals) in
              (match etype with
                 A.StructType struct_type ->
                  let field_indexing_map = StringMap.find struct_type
struct_indexing_map in
                  let index = StringMap.find field_name field_indexing_map
in
                  let struct_llvalue = lookup s in
                  let struct_field_pointer = L.build_struct_gep
struct_llvalue index "struct_field_pointer" builder in
                  ignore(L.build_store e2' struct_field_pointer builder);
e2'
                | _ -> raise (Failure("AccessStructField failed: " ^ s ^
"is not a struct type")))
          | _ -> raise (Failure("AccessStructField failed")))
        | _ -> raise (Failure("Assign failed")))
```

```
      | A.Call ("print", [e]) | A.Call ("printb", [e]) ->
            L.build_call printf_func [| int_format_str ; (expr builder e)
|] "printf" builder
      | A.Call ("printfloat", [e]) -> L.build_call printf_func [|
float_format_str; (expr builder e) |] "printf" builder
      | A.Call ("prints", [e]) ->
         L.build_call printf_func [| str_format_str; (expr builder e) |]
"printf" builder
      | A.Call ("printint", [e]) ->  L.build_call printf_func [|
int_format_str2; (expr builder e) |] "printf" builder
      | A.Call ("printstring", [e]) ->  L.build_call printf_func [|
str_format_str2; (expr builder e) |] "printf" builder

      | A.Call (f, act) ->
         let (fdef, fdecl) = StringMap.find f function_decls in
              let actuals =
            List.rev (List.map (expr builder) (List.rev act)) in
              let result = (match fdecl.A.typ with A.Void -> ""
                                                 | _ -> f ^ "_result") in
         L.build_call fdef (Array.of_list actuals) result builder

      | A.ObjectCall (g, "printGraph", []) -> let g_val = expr builder g in
        ignore(L.build_call printGraph_f [| g_val |] "" builder); g_val

      | A.ObjectCall (l, "printList", []) -> let l_val = expr builder l in
        ignore(L.build_call printList_f [| l_val |] "" builder); l_val

      | A.ObjectCall (l, "lsize", []) -> let l_val = expr builder l in
        let size_ptr = L.build_call sizeList_f [| l_val |] "" builder; in
size_ptr


      | A.ObjectCall(g, "isEmpty", []) -> let g_val = expr builder g in
        let bool_ptr = L.build_call isEmpty_f [| g_val |] "isEmpty" builder
in
        bool_ptr

      | A.ObjectCall(g, "size", []) -> let g_val = expr builder g in
        let size_ptr = L.build_call size_f [| g_val |] "isEmpty" builder in
        size_ptr

      | A.ObjectCall(g, "weight", [e1; e2]) -> let g_val = expr builder g
in
        let e1' = expr builder e1 in
        let e2' = expr builder e2 in
        let weight_ptr = L.build_call getWeight_f [| g_val; e1'; e2' |]
"getWeight" builder in
        weight_ptr
```

```
      | A.ObjectCall(n, "modifyVisited", [e]) -> let n_val = expr builder n
in
        let bool_visited = expr builder e in
        ignore(L.build_call modifyVisited_f [| n_val; bool_visited|] ""
builder); n_val

      | A.ObjectCall(g, "contains", [e]) -> let g_val = expr builder g in
        let e_val = expr builder e in
        let bool_ptr = L.build_call contains_f [| g_val; e_val |]
"contains" builder in bool_ptr

      | A.ObjectCall(g, "setData", [e]) -> let g_val = expr builder g in
        let e_val = expr builder e in
        let d_typ = L.type_of e_val in
        let d_ptr = L.build_malloc d_typ "tmp" builder in
        ignore(L.build_store e_val d_ptr builder);
        let e_val_ptr = L.build_bitcast d_ptr (L.pointer_type i8_t) "ptr"
builder in
        ignore(L.build_call setData_f [| g_val; e_val_ptr |] "" builder);
g_val

      | A.ObjectCall (q, "qadd", [e]) ->
        let q_val = expr builder q in
        let e_val = expr builder e in
        let d_ltyp = L.type_of e_val in
        let d_ptr = L.build_malloc d_ltyp "tmp" builder in
        ignore(L.build_store e_val d_ptr builder);
        let void_e_ptr = L.build_bitcast d_ptr (L.pointer_type i8_t) "ptr"
builder in
        ignore (L.build_call enqueue_f [| q_val; void_e_ptr|] "" builder);
q_val

      | A.ObjectCall (q, "qremove", []) ->
        let q_val = expr builder q in
        ignore (L.build_call dequeue_f [| q_val|] "" builder); q_val

      | A.ObjectCall (q, "qfront", []) ->
        let q_val = expr builder q in
        let n = idtostring q in
        let q_type = getQueueType (lookup_types n) in
        let val_ptr = L.build_call front_f [| q_val |] "val_ptr" builder in
        let l_dtyp = ltype_of_typ q_type in
        let d_ptr = L.build_bitcast val_ptr (L.pointer_type l_dtyp) "d_ptr"
builder in
        (L.build_load d_ptr "d_ptr" builder)

      | A.ObjectCall (q, "qsize", []) ->
        let q_val = expr builder q in
        let size_ptr = L.build_call sizeQ_f [| q_val|] "" builder in
size_ptr

      |  A.ObjectCall (p, "p_push", [e]) ->
```

```
        let pqptr = expr builder p in
        let e_val = expr builder e in
        ignore (L.build_call pushPQ_f [| pqptr; e_val |] "" builder);
        pqptr
      |  A.ObjectCall (p, "p_delete", []) ->
        let pqptr = expr builder p in
        let nodeptr = L.build_call deletePQ_f [| pqptr |] "data" builder in
        nodeptr
      | A.ObjectCall (p, "p_size", []) ->
        let pqptr = expr builder p in
        let size_ptr = L.build_call sizePQ_f [| pqptr |] "isEmpty" builder
in
        size_ptr

      | A.ObjectCall (l, "l_add", [e]) ->
        let l_ptr = expr builder l in
        let e_val = expr builder e in
        let n = idtostring l in
        let l_type = getListType (lookup_types n) in
        ( match l_type with
           A.String -> ignore(L.build_call addList_f [| l_ptr; e_val |] ""
builder); l_ptr
          | _ ->
            let d_ltyp = L.type_of e_val in
            let d_ptr = L.build_malloc d_ltyp "tmp" builder in
            ignore(L.build_store e_val d_ptr builder);
          let void_e_ptr = L.build_bitcast d_ptr (L.pointer_type i8_t)
"ptr" builder in
          ignore (L.build_call addList_f [| l_ptr ; void_e_ptr |] ""
builder);
        l_ptr


        )

      | A.ObjectCall (l, "l_delete", [e]) ->
        let l_ptr = expr builder l in
        let e_val = expr builder e in
        ignore (L.build_call delList_f [| l_ptr; e_val |] "" builder);
        l_ptr

      (* TO BE INCORPORATED WITH INT *)
      | A.ObjectCall (l, "l_get", [e]) ->
        let l_ptr = expr builder l in
        let e_val = expr builder e in
        let n = idtostring l in
        let l_type = getListType (lookup_types n) in
        let val_ptr = L.build_call getList_f [| l_ptr; e_val |] "val_ptr"
builder in
        (match l_type with
           A.String -> val_ptr
          | _ ->
```

```
            let l_dtyp = ltype_of_typ l_type in
            let d_ptr = L.build_bitcast val_ptr (L.pointer_type l_dtyp)
"d_ptr" builder in
            (L.build_load d_ptr "d_ptr" builder))

      |  A.ObjectCall(_, f, act) ->
         let (fdef, fdecl) = StringMap.find f function_decls in
         let actuals =
            List.rev (List.map (expr builder) (List.rev act)) in
         let result = (match fdecl.A.typ with A.Void -> ""
                                            | _ -> f ^ "_result") in
         L.build_call fdef (Array.of_list actuals) result builder

      | A.Infinity -> L.const_int i32_t (1000000)
      | A.NegInfinity -> L.const_int i32_t (-1000000)


    in

    (* Invoke "f builder" if the current block doesn't already
       have a terminal (e.g., a branch). *)
    let add_terminal builder f =
      match L.block_terminator (L.insertion_block builder) with
            Some _ -> ()
      | None -> ignore (f builder) in

    (* Build the code for the given statement; return the builder for
       the statement's successor *)
    let rec stmt builder = function
            A.Block sl -> List.fold_left stmt builder sl
      | A.Expr e -> ignore (expr builder e); builder
      | A.Return e -> ignore (match fdecl.A.typ with
                A.Void -> L.build_ret_void builder
              | _ -> L.build_ret (expr builder e) builder); builder
      | A.If (predicate, then_stmt, else_stmt) ->
         let bool_val = expr builder predicate in
              let merge_bb = L.append_block context "merge" the_function in
              let then_bb = L.append_block context "then" the_function in
              add_terminal (stmt (L.builder_at_end context then_bb)
then_stmt)
                      (L.build_br merge_bb);
              let else_bb = L.append_block context "else" the_function in
              add_terminal (stmt (L.builder_at_end context else_bb)
else_stmt)
                           (L.build_br merge_bb);
              ignore (L.build_cond_br bool_val then_bb else_bb builder);
              L.builder_at_end context merge_bb
      | A.While (predicate, body) ->
               let pred_bb = L.append_block context "while" the_function in
               ignore (L.build_br pred_bb builder);
```

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

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

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

      | A.For (e1, e2, e3, body) -> stmt builder
                  ( A.Block [A.Expr e1 ;
                        A.While (e2, A.Block [body ;
                                              A.Expr e3]) ] )
    in

    (* Build the code for each statement in the function *)
    let builder = stmt builder (A.Block fdecl.A.body) in

    (* Add a return if the last block falls off the end *)
    add_terminal builder (match fdecl.A.typ 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
```

semant.ml

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

open Ast


module StringMap = Map.Make(String)

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

   Check each global variable, then check each function *)

let check (globals, functions, structs) =

  (* Raise an exception if the given list has a duplicate *)
  let report_duplicate exceptf list =
    let rec helper = function
           n1 :: n2 :: _ when n1 = n2 -> raise (Failure (exceptf n1))
      | _ :: t -> helper t
```

```
      | [] -> ()
    in helper (List.sort compare list)
  in

  (* Raise an exception if a given binding is to a void type *)
  let check_not_void exceptf = function
      (Void, n) -> raise (Failure (exceptf n))
    | _ -> ()
  in

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




  (**** Checking Global Variables ****)

  List.iter (check_not_void (fun n -> "illegal void global " ^ n)) globals;

  report_duplicate (fun n -> "duplicate global " ^ n) (List.map snd
globals);




  (**** Checking Structs ****)
  report_duplicate (fun n -> "duplicate struct type " ^ n)
    (List.map (fun sd -> sd.sname) structs);




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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

(* Function declaration for a named function *)
let built_in_decls =  StringMap.add "print"
   { typ = Void; fname = "print"; formals = [(Int, "x")];
     locals = []; body = [] }

     (StringMap.add "printint"
  { typ = Void; fname = "printint"; formals = [(Int, "x")];
     locals = []; body = [] }

     (StringMap.add "printfloat"
   { typ = Void; fname = "printfloat"; formals = [(Float, "x")];
     locals = []; body = [] }

     (StringMap.add "printb"
   { typ = Void; fname = "printb"; formals = [(Bool, "x")];
     locals = []; body = [] }

     (StringMap.add "prints"
   { typ = Void; fname = "prints"; formals = [(String, "x")];
     locals = []; body = [] }

    (StringMap.add "printstring"
   { typ = Void; fname = "printstring"; formals = [(String, "x")];
     locals = []; body = [] }

     (StringMap.add "printbig"
   { typ = Void; fname = "printbig"; formals = [(Int, "x")];
     locals = []; body = [] }

    (StringMap.add "modifyVisited"
   { typ = Void; fname = "modifyVisited"; formals = [(Bool, "x")];
     locals = []; body = [] }

    (StringMap.add "printGraph"
   { typ = Void; fname = "printGraph"; formals = [];
     locals = []; body = [] }

    (StringMap.add "printList"
   { typ = Void; fname = "printList"; formals = [];
     locals = []; body = [] }

    (StringMap.add "isEmpty"
   { typ = Bool; fname = "isEmpty"; formals = [];
```

```
       locals = []; body = [] }

      (StringMap.add "size"
     { typ = Int; fname = "size"; formals = [];
       locals = []; body = [] }

      (StringMap.add "lsize"
     { typ = Int; fname = "lsize"; formals = [];
       locals = []; body = [] }

      (StringMap.add "weight"
     { typ = Int; fname = "weight"; formals = [(AnyType, "x"); (AnyType,
"x")];
       locals = []; body = [] }

      (StringMap.add "contains"
     { typ = Bool; fname = "isEmpty"; formals = [(String, "x")];
       locals = []; body = [] }

      (StringMap.add "setData"
     { typ = Void; fname = "setData"; formals = [(AnyType, "x")];
       locals = []; body = [] }

       (StringMap.add "p_push"
     { typ = Void; fname = "p_push"; formals = [(AnyType, "x")];
       locals = []; body = [] }

       (StringMap.add "p_delete"
     { typ = AnyType; fname = "p_delete"; formals = [];
       locals = []; body = [] }

       (StringMap.add "p_size"
     { typ = Int; fname = "p_size"; formals = [];
       locals = []; body = [] }

       (StringMap.add "qremove"
     { typ = Void; fname = "qremove"; formals = [];
       locals = []; body = [] }

       (StringMap.add "qadd"
     { typ = QueueType(AnyType); fname = "qadd"; formals = [(AnyType,
"x")];
       locals = []; body = [] }

       (StringMap.add "qfront"
     { typ = AnyType; fname = "qfront"; formals = [];
       locals = []; body = [] }

      (StringMap.add "qsize"
     { typ = Int; fname = "qsize"; formals = [];
       locals = []; body = [] }
```

```
      (StringMap.add "l_add"
    { typ = Void; fname = "l_add"; formals = [(AnyType, "x")];
      locals = []; body = [] }

      (StringMap.add "l_delete"
    { typ = Void; fname = "l_delete"; formals = [(Int, "x")];
      locals = []; body = [] }

      (StringMap.singleton "l_get"
    { typ = AnyType; fname = "l_get"; formals = [(Int, "x")];
      locals = []; body = [] }

    ))))))))))))))))))))))))


   in

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

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

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



  let check_AccessStructField struct_name field_name =
    match struct_name with
        StructType struct_type ->
          (let the_struct_type = try List.find (fun s -> s.sname =
struct_type) structs
                                 with Not_found -> raise (Failure("struct
type " ^ struct_type ^ " is undefined")) in
          try fst( List.find (fun s -> snd(s) = field_name)
the_struct_type.sformals)
          with Not_found -> raise (Failure("struct " ^ struct_type ^ " does
not contain field " ^ field_name)))
      | _ -> raise (Failure(string_of_typ struct_name ^ " is not a struct
type"))
  in



  let check_function func =

    List.iter (check_not_void (fun n -> "illegal void formal " ^ n ^
      " in " ^ func.fname)) func.formals;
```

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

    List.iter (check_not_void (fun n -> "illegal void local " ^ n ^
      " in " ^ func.fname)) func.locals;

    report_duplicate (fun n -> "duplicate local " ^ n ^ " in " ^
func.fname)
      (List.map snd func.locals);

    (* Type of each variable (global, formal, or local *)
    let symbols = List.fold_left (fun m (t, n) -> StringMap.add n t m)
                                     StringMap.empty
                                (globals @ func.formals @ func.locals )
    in

    let type_of_identifier s =
      try StringMap.find s symbols
      with Not_found -> raise (Failure ("undeclared identifier " ^ s))
    in

    let getQueueType = function
       QueueType(typ) -> typ
      | _ -> Void
    in

     let getNodeType = function
       NodeType(typ) -> typ
      | _ -> Void
    in




    let getListType = function
       ListType(typ) -> typ
      | _ -> String
    in

    (* Return the type of an expression or throw an exception *)
    let rec expr = function
            IntLit _ -> Int
      | Infinity -> Int
      | NegInfinity -> Int
      | BoolLit _ -> Bool
      | FloatLit _ -> Float
      | StringLit _ -> String
      | Queue (t, _) -> QueueType(t)
      | List (t,_) -> ListType(t)
```

```
      | PQueue _ -> PQueueType
      | Node(_, t) -> NodeType(t)
      | Id s -> type_of_identifier s
      | Graph(t) -> GraphType(t)
      | Binop(e1, op, e2) as e -> let t1 = expr e1
                                  and t2 = expr e2 in
           (match op with
           Add       when t1 = Float && t2 = Float -> Float
         | Add       when t1 = Int && t2 = Int -> Int
         | And       when t1 = Bool && t2 = Bool -> Bool
         | Div       when t1 = Float && t2 = Float -> Float
         | Div       when t1 = Int && t2 = Int -> Int
         | Equal     when t1 = t2 -> Bool
         | Geq       when t1 = Float && t2 = Float -> Bool
         | Geq       when t1 = Int && t2 = Int -> Bool
         | Greater   when t1 = Float && t2 = Float -> Bool
         | Greater   when t1 = Int && t2 = Int -> Bool
         | Leq       when t1 = Float && t2 = Float -> Bool
         | Leq       when t1 = Int && t2 = Int -> Bool
         | Less      when t1 = Float && t2 = Float -> Bool
         | Less      when t1 = Int && t2 = Int -> Bool
         | Mult      when t1 = Float && t2 = Float -> Float
         | Mult      when t1 = Int && t2 = Int -> Int
         | Neq    when t1 = t2 -> Bool
         | Or        when t1 = Bool && t2 = Bool -> Bool
         | Sub       when t1 = Float && t2 = Float -> Float
         | Sub       when t1 = Int && t2 = Int -> Int
         | _ -> raise (Failure ("illegal binary operator " ^
           string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
           string_of_typ t2 ^ " in " ^ string_of_expr e)))

      | Unop(op, e) as ex -> let t = expr e in
           (match op with
            Neg when t = Int -> Int
          | Not when t = Bool -> Bool
       | _ -> raise (Failure ("illegal unary operator " ^ string_of_uop op
^
                          string_of_typ t ^ " in " ^ string_of_expr ex)))

      | AccessStructField(e, field_name) -> let lt = expr e in
         check_AccessStructField lt field_name

      | Noexpr -> Void

      | Assign(e1, e2) as ex ->
        (match e1 with
          Id s -> let lt = type_of_identifier s and rt = expr e2 in
            if rt <> AnyType then check_assign lt rt (Failure ("illegal
assignment " ^ string_of_typ lt ^
                                    " = " ^ string_of_typ rt ^ " in " ^
string_of_expr ex))
```

```
            else  check_assign rt rt (Failure ("illegal assignment " ^
string_of_typ lt ^
                                    " = " ^ string_of_typ rt ^ " in " ^
                                    string_of_expr ex))
        | AccessStructField(_, _) -> expr e2
        | _ -> raise (Failure("illegal graph operator")))

      | GraphOp(s1, gop, s2) -> let t1 = type_of_identifier s1 and t2 =
type_of_identifier s2 in
        (match t1 with
            GraphType(typ) ->
            (match gop with
              AddNode when t1 = GraphType typ && t2 = String -> GraphType
typ
            | RemoveNode when t1 = GraphType typ && t2 = String ->
GraphType typ
            | _ -> raise (Failure("illegal graph operator"))
            )
          | _ -> raise(Failure("operand is not a graph"))
        )

      | GraphOpAddEdge(e, _, gop2, s1, s2) -> let t1 = expr e and
       t2 = type_of_identifier s1 and t3 = type_of_identifier s2 in

        (match t1 with
            GraphType(typ) ->
            (match gop2 with
              AddEdge when t2 = NodeType typ && t3 = NodeType typ ->
GraphType typ
              | _ -> raise(Failure("illegal graph operator"))
            )
          | _ -> raise(Failure("operand is not a graph"))
        )

       | GraphOpRemoveEdge(e, gop3, s1, s2) -> let t1 = type_of_identifier
e and
       t2 = type_of_identifier s1 and t3 = type_of_identifier s2 in
        (match t1 with
            GraphType(typ) ->
            (match gop3 with
              RemoveEdge when t2 = NodeType typ && t3 = NodeType typ ->
GraphType typ
              | _ -> raise(Failure("illegal graph operator"))
            )
          | _ -> raise(Failure("operand is not a graph"))
        )


      | NodeOp(e, nop, s) -> let t1 = expr e and t2 = type_of_identifier s
in

        (match nop with
```

```
          AccessNode ->
          (match t1 with
           GraphType typ when t1 = GraphType(typ) && t2 = String ->
NodeType(typ)
           | _ -> raise (Failure("not a graph"))
          )
        )

      | NodeOp2(e, nop2) -> let t1 = expr e in
        (match t1 with
          NodeType typ ->
          (match nop2 with
             GetName -> String
           | GetData -> typ
           | GetVisited -> Bool
           | GetinNodes -> ListType(String)
           | GetoutNodes -> ListType(String)
          )
          | _ -> raise(Failure("Not a vaild node operator"))
        )


      | Call(fname, actuals) as call -> let fd = function_decl fname in
         if List.length actuals != List.length fd.formals then
           raise (Failure ("expecting " ^ string_of_int
             (List.length fd.formals) ^ " arguments in " ^ string_of_expr
call))
         else
           List.iter2 (fun (ft, _) e -> let et = expr e in
              ignore (check_assign ft et
                (Failure ("illegal actual argument found " ^ string_of_typ
et ^
                " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr
e))))
             fd.formals actuals;
           fd.typ

      | ObjectCall(oname, fname, actuals) as objectcall -> let fd =
function_decl fname in
        let returntype = ref (fd.typ) in
        if List.length actuals != List.length fd.formals then
          raise (Failure ("expecting " ^ string_of_int
             (List.length fd.formals) ^ " arguments in " ^ string_of_expr
objectcall))

        else
           List.iter2 (fun (ft, _) e -> let et = expr e in

            (* if fname = "qfront" then let _ = print_endline
(string_of_typ actqtype) in returntype := actqtype *)
              if fname = "qadd" then
                 let acttype = expr oname in
```

```
                let actqtype = getQueueType acttype in
               ignore(check_assign actqtype et (Failure ("illegal actual
queue argument found " ^ string_of_typ et ^
                " expected " ^ string_of_typ actqtype ^ " in " ^
string_of_expr e)))

             else if fname = "weight" then
                let acttype = expr (List.hd actuals) in
                 ignore(check_assign acttype et (Failure ("illegal actual
node argument found " ^ string_of_typ et ^
                " expected " ^ string_of_typ acttype ^ " in " ^
string_of_expr e)))

             else if fname = "p_push" then
                let acttype = expr (List.hd actuals) in
                 ignore(check_assign acttype et (Failure ("illegal actual
pqueue argument found " ^ string_of_typ et ^
                " expected " ^ string_of_typ acttype ^ " in " ^
string_of_expr e)))

             else if fname = "setData" then
                let acttype = expr oname in
                let actntype = getNodeType acttype in
                 ignore(check_assign actntype et (Failure ("illegal actual
node argument found " ^ string_of_typ et ^
                " expected " ^ string_of_typ actntype ^ " in " ^
string_of_expr e)))
             else if fname = "l_add" then
               let acttype = expr oname in
               let actltype = getListType acttype in
               ignore(check_assign actltype et (Failure ("illegal actual
list argument found " ^ string_of_typ et ^
                " expected " ^ string_of_typ actltype ^ " in " ^
string_of_expr e)))
             else ignore (check_assign ft et (Failure ("illegal actual
argument found " ^ string_of_typ et ^
                " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr
e)))) fd.formals actuals;
           !returntype



    in

    let check_bool_expr e = if expr e != Bool
                            then raise (Failure ("expected Boolean
expression in " ^ string_of_expr e))
                            else () in

    (* Verify a statement or throw an exception *)
    let rec stmt = function
            Block sl -> let rec check_block = function
```

```
          [Return _ as s]  -> stmt s
        | Return _ :: _    -> raise (Failure "nothing may follow a
return")
        | Block sl :: ss   -> check_block (sl @ ss)
        | s :: ss          -> stmt s ; check_block ss
        | [] -> ()
       in check_block sl
     | Expr e -> ignore (expr e)
     | Return e -> let t = expr e in if t = func.typ then () else
       raise (Failure ("return gives " ^ string_of_typ t ^ " expected " ^
                       string_of_typ func.typ ^ " in " ^ string_of_expr
e))
     | If(p, b1, b2) -> check_bool_expr p; stmt b1; stmt b2
     | For(e1, e2, e3, st) -> ignore (expr e1); check_bool_expr e2;
                              ignore (expr e3); stmt st
     | While(p, s) -> check_bool_expr p; stmt s
   in

   stmt (Block func.body)

  in
  List.iter check_function functions
```

queue.h

```
#ifndef __QUEUE_H__
#define __QUEUE_H__

struct Node {
       void *data;
       struct Node *next;
};

struct QueueId {
       struct Node *front;
       struct Node *rear;
       int size;
};

struct QueueId *initQueueId();
void enqueue(struct QueueId *queue, void *data);
void dequeue(struct QueueId *queue);
void *front(struct QueueId *queue);
int q_size(struct QueueId* queue);

#endif /* #ifndef _QUEUE_H_ */
```

queue.c

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

```
#include <stdlib.h>
#include "queue.h"


int q_size(struct QueueId* queue) {
	return queue->size;
}

struct QueueId* initQueueId() {
	struct QueueId* new = (struct QueueId*) malloc(sizeof(struct
QueueId));
	new->front = 0;
	new->rear = 0;
	new->size = 0;
	return new;
}

void enqueue(struct QueueId *queue, void *data) {
	struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
	temp -> data = data;
	temp -> next = NULL;
	if (queue->front == NULL && queue->rear == NULL) {
		queue->front = queue->rear = temp;
		queue->size++;
		return;
	}
	queue->rear->next = temp;
	queue->rear = temp;
	queue->size++;
}

void dequeue(struct QueueId *queue) {
	struct Node* temp = queue->front;
	if (queue->front == NULL) {
		return;
	}
	if (queue->front==queue->rear) {
		queue->front = queue->rear = NULL;
	}
	else {
		queue->front = queue->front->next;
	}
	queue->size--;
	free(temp);
}

void *front(struct QueueId *queue) {
	if(queue->front == NULL) {
		return NULL;
	}
	return queue->front->data;
}
```

pqueue.h

```
#include "linkedlist.h"
#include "map.h"
#include "node.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifndef __PQUEUE_H__
#define __PQUEUE_H__


extern struct pqueue *pq_init();

extern void pq_push(struct pqueue *pq, struct node *n);

extern struct node *pq_delete(struct pqueue *pq);

extern bool is_empty(struct pqueue *pq);

extern int p_size(struct pqueue *pq);



#endif
```

pqueue.c

```
#include "linkedlist.h"
#include "map.h"
#include "pqueue.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int pq_size;
int initial_pq_size = 8;
int count;


struct pqueue {
	struct node **node_arr;
	int pq_size;
	int count;
};



struct pqueue *pq_init() {
	struct pqueue *pq = malloc(sizeof(struct pqueue));
```

```
	pq->count = 0;
	pq->pq_size = initial_pq_size;
	pq->node_arr = (struct node **) malloc(sizeof(struct node *) *
initial_pq_size);
	return pq;
}


void pq_push(struct pqueue *pq, struct node *n)
{
	// Repq_size the heap if necessary
	if (pq->count == pq->pq_size)
	{
		pq->pq_size *= 2;
		pq->node_arr = realloc(pq->node_arr, sizeof(struct node *) *
pq->pq_size);
	}

	int new_node_data = *((int *) n->data);

	int index = pq->count;
	pq->count++;

	// Find the correct queue index to insert the element
	while (index > 0) {

		int pq_node_data = *((int *) pq->node_arr[index - 1]->data);

		if (new_node_data <= pq_node_data) {
			break;
		}
		else {
			index--;
		}
	}

	// Shift the queue elements to the right of index by 1 space
	for (int i = (pq->count) - 1; i >= index; i--) {
		pq->node_arr[i] = pq->node_arr[i - 1];
	}

	pq->node_arr[index] = n;
}


struct node *pq_delete(struct pqueue *pq)
{
	struct node *removed = pq->node_arr[--pq->count];
	return removed;
}
```

```
bool is_empty(struct pqueue *pq) {
	return pq->count == 0;
}


int p_size(struct pqueue *pq) {
	return pq->count;
}
```

node.h

```
#include "map.h"
#include "linkedlist.h"
#include <stdbool.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>


struct node {
	char *name;
	bool visited;
	struct map *inNodes;
	struct map *outNodes;
	void *data;
};


struct node *n_init(char *name);

char *get_name(struct node *curr_node);

void set_data(struct node *curr_node, void *data);


bool get_visited(struct node *curr_node);

void modify_visited(struct node *curr_node, bool val);

struct List *get_inNodes(struct node *curr_node);

struct List *get_outNodes(struct node *curr_node);

void *get_data(struct node *curr_node);

void print_node(struct node *node);
```

node.c

```
#include "map.h"
#include "node.h"
#include <stdbool.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
//n_init added

struct node *n_init(char *name) {
	struct node *new_node = malloc(sizeof(struct node));
	new_node->name = name;
	new_node->visited = false;
	new_node->inNodes = m_init();
	new_node->outNodes = m_init();
	new_node->data = malloc(sizeof(void *));
	return new_node;
}

void set_data(struct node *curr_node, void *data) {
	curr_node->data = data;
}


char *get_name(struct node *curr_node) {
	return curr_node->name;

}

bool get_visited(struct node *curr_node) {
	return curr_node->visited;

}

void modify_visited(struct node *curr_node, bool val){
	curr_node->visited = val;
	return;
}

struct List *get_inNodes(struct node *curr_node) {
	struct List *l = l_init();
	for (int i = 0; i < curr_node->inNodes->size; i++) {

		char *key = m_key(curr_node->inNodes, i);
		l_add(l, key);
	}
	return l;
}
```

```
struct List *get_outNodes(struct node *curr_node) {
	struct List *l = l_init();
	for (int i = 0; i < curr_node->outNodes->size; i++) {
		char *key = m_key(curr_node->outNodes, i);
		l_add(l, key);
	}
	return l;
}

void *get_data(struct node *curr_node) {
	return curr_node->data;
}

void print_node(struct node *n) {
	printf("%s ", "node name: ");
	printf("%s\n", n->name);
	printf("%s ", "node visited: ");
	printf("%d\n", n->visited);
	printf("%s ", "node inNodes: ");
	print_map(n->inNodes);
	printf("%s ", "node outNodes: ");
	print_map(n->outNodes);

}
```

map.h

```
#ifndef __MAP_H__
#define __MAP_H__

#define INFINITY 50000
struct map {
	int table_size;
	int size;
	char **nodes;
	int *weight;
};

extern struct map *m_init();

extern void m_insert(struct map *curr_map, char *node, int val);

extern int m_get(struct map *curr_map, char *node);

extern char *m_key(struct map *curr_map, int index);

extern void m_remove(struct map *curr_map, char *node);

extern int m_size(struct map *curr_map);

extern void m_free(struct map *curr_map);
```

```
extern void print_map(struct map* curr_map);


#endif
```

map.c

```
#include "map.h"

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



struct map *m_init() {
	int INITIAL_SIZE = 3;
	struct map *new_map = malloc(sizeof(struct map));
	new_map->nodes = malloc(sizeof(char *) * INITIAL_SIZE);
	for (int i = 0; i < INITIAL_SIZE; i++) {
		new_map->nodes[i] = NULL;
	}
	new_map->weight = malloc(sizeof(int) * INITIAL_SIZE);
	for (int i = 0; i < INITIAL_SIZE; i++) {
		new_map->weight[i] = 0;
	}
	new_map->table_size = INITIAL_SIZE;
	new_map->size = 0;
	return new_map;
}

void m_insert(struct map *curr_map, char *node, int val) {

	if(m_get(curr_map, node) != INFINITY) {
		return;
	}
	if (curr_map->size == curr_map->table_size) {
		curr_map->table_size = curr_map->table_size * 2;
		curr_map->nodes = realloc(curr_map->nodes, sizeof(char *) *
curr_map->table_size);
		curr_map->weight = realloc(curr_map->weight, sizeof(int) *
curr_map->table_size);
	}
	//curr_map->nodes[curr_map->size] = malloc(sizeof(char *));
	//curr_map->weight[curr_map->size] = malloc(sizeof(int));

	curr_map->nodes[curr_map->size] = node;
	curr_map->weight[curr_map->size] = val;
	curr_map->size += 1;


}
```

```
extern char *m_key(struct map *curr_map, int index) {
	for (int i = 0; i < curr_map->size; i++) {
		if (i == index) {
			return curr_map->nodes[i];
		}
	}
	return NULL;
}

int m_get(struct map *curr_map, char *node) {
	for (int i = 0; i < curr_map->size; i++) {
		if (strcmp(curr_map->nodes[i], node) == 0) {
			return curr_map->weight[i];
		}
	}
	return INFINITY;
}

void m_remove(struct map *curr_map, char *node) {
	int index = curr_map->size;
	for (int i = 0; i < curr_map->size; i++) {
		if (strcmp(curr_map->nodes[i], node) == 0) {
			for (int j = i; j < curr_map->size - 1; j++) {
				curr_map->nodes[j] = curr_map->nodes[j+1];
				curr_map->weight[j] = curr_map->weight[j+1];
			}
			curr_map->nodes[curr_map->size] = NULL;
			curr_map->weight[curr_map->size] = 0;
			curr_map->size = curr_map->size - 1;
		}
	}

}

int m_size(struct map *curr_map) {
	return curr_map->size;
}

void m_free(struct map *curr_map) {
	struct map *m = (struct map *) curr_map;
	//for (int i = 0; i < curr_map->size; i++) {
		//free(curr_map->nodes[i]);
		//free(curr_map->weight[i]);
	//}
	free(m->nodes);
	free(m->weight);
	free(m);
}

void print_map(struct map *curr_map) {
	printf("%s", "{");
```

```
	for (int i = 0; i < curr_map->size; i++) {
		printf("%s", "(");

		char *nod = curr_map->nodes[i];
		int w = curr_map->weight[i];
		printf("%s", nod);
		printf("%s", ", ");
		printf("%d", w);
		printf("%s", ")");
		printf("%s", ", ");
	}
	printf("%s", "}\n");

}
```

linkedlist.h

```
#ifndef __LINKEDLIST_H__
#define __LINKEDLIST_H__

struct List {
	struct ListNode *head;
	int size;
};

struct ListNode {
	void *data;
	struct ListNode *next;
};

struct List* l_init();

void l_add(struct List *list, void *data);

void l_delete(struct List *list, int index);

void* l_get(struct List *list, int index);

struct ListNode* l2_get(struct List *list, int index);

int l_isEmpty(struct List *list);

int l_size(struct List *list);

void print_list(struct List *list);


#endif
```

linkedlist.c

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

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

void l_add(struct List *list, void *data) {
	struct ListNode* new = (struct ListNode*) malloc(sizeof(struct
ListNode));
	if (new == NULL)
		return;

	new->data = data;
	new->next = NULL;
	if (list->head == NULL) {
		list->head = new;
		list->size++;
	} else {
		struct ListNode* temp = list->head;
		while (temp->next != NULL) {
			temp = temp->next;
		}
		temp->next = new;
		list->size++;
	}
}
void l_delete(struct List *list, int index) {
	if (l_isEmpty(list)) {
		return;
	}
	if (index == 0) {
		struct ListNode* temp = list->head;
		list->head = list->head->next;
		free(temp);
		list->size--;
		return;
	}

	int ctr = 0;
	struct ListNode* temp = list->head;
	while(ctr < index-1) {
		temp = temp->next;
		ctr++;
	}
	temp->next = temp->next->next;
```

```
	temp = temp->next;
	free(temp);
	list->size--;
}

void* l_get(struct List *list, int index) {
	struct ListNode* temp = list->head;
	while (index > 0) {
		temp = temp->next;
		index--;
	}
	return temp->data;
}

struct ListNode* l2_get(struct List *list, int index) {
	struct ListNode* temp = list->head;
	while (index > 0) {
		temp = temp->next;
		index--;
	}
	return temp;
}


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

int l_size(struct List *list) {
	return (list->size);
}

void print_list(struct List *list) {
	printf("%s", "{");
	for (int i = 0; i < list->size; i++) {
		char *nod = l_get(list, i);
		printf("%s", nod);
		printf("%s", ", ");
	}
	printf("%s", "}\n");

}
```

graph.h

```
#include "node.h"
#include "map.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "linkedlist.h"
```

```
#ifndef __GRAPH_H__
#define __GRAPH_H__


struct graph {
	struct List *nodes;
	int size;
};


extern struct graph *g_init();

extern void addNode (struct graph* g, struct node *n);

extern void removeNode (struct graph* g, struct node *n);

extern void addEdge(struct graph* g, struct node *n1, struct node *n2, int
weight);

extern void removeEdge(struct graph* g, struct node *n1, struct node *n2);

extern int getWeight(struct graph* g, struct node *n1, struct node *n2);

extern struct node *indexNode(struct graph* g, int index);

void freeGraph(struct graph* g);

void removeAllNodes(struct graph* g);

void printGraph(struct graph* g);

bool isEmpty(struct graph* g);

int size(struct graph* g);

bool contains(struct graph* g, char *name);

struct node *getNode(struct graph* g, char *name);



#endif
```

graph.c

```
#include "map.h"
#include "graph.h"
#include "linkedlist.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
```

```
//g_init, addNode added

extern struct graph *g_init() {
	struct graph *new_graph = malloc(sizeof(struct graph));
	struct List *new_list = l_init();
	new_graph->nodes = new_list;
	new_graph->size = 0;
	return new_graph;
}

extern void addNode (struct graph* g, struct node *n) {
	for (int i = 0; i < g->size; i++) {
		struct node *no = (struct node *)((l2_get(g->nodes, i))-
>data);
		if (no->name == n->name) {
			return;
		}
	}
	l_add(g->nodes, n);
	g->size++;
}

extern void removeNode (struct graph* g, struct node *n) {
	for (int i = 0; i < g->size; i++) {
		struct node *no = (struct node *)((l2_get(g->nodes, i))-
>data);
		if (strcmp(no->name, n->name) == 0) {
			struct ListNode *listnode = l2_get(g->nodes, i);
			l_delete(g->nodes, i);
			g->size--;
		}
	}

	for (int i = 0; i < g->size; i++) {
		struct node *no = (struct node *)((l2_get(g->nodes, i))-
>data);
		if (m_get(no->inNodes, n->name) != -1) {
			m_remove(no->inNodes, n->name);
		}
		if (m_get(no->outNodes, n->name) != -1) {
			m_remove(no->outNodes, n->name);
		}
	}
}

extern void addEdge(struct graph* g, struct node *n1, struct node *n2, int
weight) {
	// check that n1 and n2 exist
	bool n1exist = false;
	int n1index = -1;
	bool n2exist = false;
	int n2index = -1;
```

```
	for (int i = 0; i < g->size; i++) {
		struct node *no = (struct node *)(l2_get(g->nodes, i))->data;
		if (no->name == n1->name) {
			n1exist = true;
			n1index = i;
		}
		if (no->name == n2->name) {
			n2exist = true;
			n2index = i;
		}
	}
	if (n1exist && n2exist != true) {
		return;
	}

	struct node *no1 = (struct node *)(l2_get(g->nodes, n1index))->data;
	struct node *no2 = (struct node *)(l2_get(g->nodes, n2index))->data;


	// add to outNodes of n1
	m_insert(no1->outNodes, n2->name, weight);

	// add to inNodes of n1
	m_insert(no2->inNodes, n1->name, weight);

}

extern void removeEdge(struct graph* g, struct node *n1, struct node *n2) {
	// check that n1 and n2 exist
	bool n1exist = false;
	int n1index = -1;
	bool n2exist = false;
	int n2index = -1;
	for (int i = 0; i < g->size; i++) {
		struct node *no = (struct node *)(l2_get(g->nodes, i))->data;
		if (no->name == n1->name) {
			n1exist = true;
			n1index = i;
		}
		if (no->name == n2->name) {
			n2exist = true;
			n2index = i;
		}
	}
	if (n1exist && n2exist != true) {
		return;
	}

	struct node *no1 = (struct node *)(l2_get(g->nodes, n1index))->data;
	struct node *no2 = (struct node *)(l2_get(g->nodes, n2index))->data;
```

```
	//check outNodes of n1
	int weight1 = m_get(no1->outNodes, no2->name);
	int weight2 = m_get(no1->outNodes, no2->name);
	if(weight1 != -1 && weight1 == weight2) {
		m_remove(no1->outNodes, no2->name);
		m_remove(no2->inNodes, no1->name);
	}
}


extern int getWeight(struct graph* g, struct node *n1, struct node *n2) {
	struct map *m = n1->outNodes;
	int result = m_get(m, n2->name);
	if (result != INFINITY) {
		return result;
	}
	else return INFINITY;

}


extern struct node *indexNode(struct graph* g, int index) {
	struct ListNode *listnode = l2_get(g->nodes, index);
	return listnode->data;
}

void removeAllNodes(struct graph* g) {
	for (int i = 0; i < g->size; i++) {
		struct ListNode *listnode = l2_get(g->nodes, i);
		struct node *no = listnode->data;
		removeNode(g, no);
	}
}


void freeGraph(struct graph* g) {
	for (int i = 0; i < g->size; i++) {
		struct ListNode *listnode = l2_get(g->nodes, i);
		free(listnode->data);
		free(listnode);
	}
	free(g->nodes);
	free(g);
}

void printGraph(struct graph* g) {
	printf("%s\n", "graph: ");
	for (int i = 0; i < g->size; i++) {
		struct ListNode *listnode = l2_get(g->nodes, i);
		struct node *nod = (struct node *)listnode->data;
		print_node(nod);
		printf("%s", "\n");
```

```
	}

}


bool isEmpty(struct graph* g) {
	if (g->size == 0) {
		return true;
	}
	else return false;
}

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

bool contains(struct graph* g, char *name) {
	for (int i = 0; i < g->size; i++) {
		struct node *no = (struct node *)(l2_get(g->nodes, i))->data;
		if (no->name == name ){
			return true;
		}
	}
	return false;

}

struct node *getNode(struct graph* g, char *name) {
	for (int i = 0; i < g->size; i++) {
		struct node *no = (struct node *)(l2_get(g->nodes, i))->data;
		if (no->name == name ){
			return no;
		}
	}
	return NULL;

}
```

BFS.yg

```
void BFS(graph<int> g1, node<int> source) {
	Queue<node<int>> p;
	list<string> listtemp;
	node<int> temp;
	node<int> temp2;
	int i;
	string tmpstring;

	p = new Queue<node<int>>();
	/* set distance of source vertex to 0 */
```

```
	source.modifyVisited(true);
	p.qadd(source);

	prints("BFS traversal of the graph: ");
	while(p.qsize() != 0) {
		temp = p.qfront();
		printstring(temp@name);
		p.qremove();
		listtemp = temp@outNodes;
		for (i = 0; i < listtemp.lsize(); i = i + 1) {
			tmpstring = listtemp.l_get(i);
			temp2 = g1~_tmpstring;
			if(temp2@visited == false) {
				temp2.modifyVisited(true);
				p.qadd(temp2);
			}
		}
		if (p.qsize() != 0) {
			printstring (" -> ");
		}
	}
}

int main() {

	/* create an undirected graph g1 */
	graph<int> g1;
	string a1;
	string a2;
	string a3;
	string a4;

	int i;

	node<int> a;
	node<int> b;
	node<int> c;
	node<int> d;

	a1 = "a";
	a2 = "b";
	a3 = "c";
	a4 = "d";

	/* add nodes to g1 */
	g1 = new graph<int>();
	g1~+a1;
	g1~+a2;
	g1~+a3;
	g1~+a4;

	a = g1~_a1;
```

```
        b = g1~_a2;
        c = g1~_a3;
        d = g1~_a4;

        g1[1]->(a, b);
        g1[2]->(a, c);

        g1[1]->(b, a);
        g1[10]->(b, d);

        g1[2]->(c, a);
        g1[5]->(c, d);

        g1[10]->(d, b);
        g1[5]->(d, c);

        BFS(g1, a);


}
```

dijkstra.yg

```
void dijkstra(graph<struct nodedata> g1, node<struct nodedata> source) {
        pqueue p;
        struct nodedata sourcedata;
        struct nodedata sourcedata2;
        node <struct nodedata> temp;
        node <struct nodedata> temp2;
        string tmpstring;
        list<string> listtemp;
        int i;

        listtemp = new list<string>();
        p = new pqueue();
        tmpstring = "temporary string";

        /* set distance of source vertex to 0 */
        sourcedata = source@data;
        sourcedata~dist = 0;
        source.setData(sourcedata);
        p.p_push(source);

        prints("Vertex              Distance from source");
    while(p.p_size() != 0) {
              temp = p.p_delete();
              printstring(temp@name);
              printstring("                        ");
              sourcedata = temp@data;
              printint(sourcedata~dist);
              listtemp = temp@outNodes;
```

```
		print(listtemp.lsize());
		for (i = 0; i < listtemp.lsize(); i = i + 1) {
			tmpstring = listtemp.l_get(i);
			temp2 = g1~_tmpstring;
			sourcedata2 = temp2@data;
			if (sourcedata2~dist > sourcedata~dist +
g1.weight(temp, temp2)) {
				sourcedata2~dist = sourcedata~dist +
g1.weight(temp, temp2);
				temp2.setData(sourcedata2);
				p.p_push(temp2);
			}
		}
	}


}


struct nodedata {
	int dist;
	node<struct nodedata> parent;
}

int main() {

	list<string> listtemp;
	list<string> listtemp2;
	/* create an undirected graph g1 */
	graph<struct nodedata> g1;
	string a1;
	string a2;
	string a3;
	string a4;

	string stringtmp;

	int i;

	struct nodedata n1;
	struct nodedata n2;
	struct nodedata n3;
	struct nodedata n4;

	node<struct nodedata> a;
	node<struct nodedata> b;
	node<struct nodedata> c;
	node<struct nodedata> d;

	n1~dist = INFINITY;
	n2~dist = INFINITY;
	n3~dist = INFINITY;
```

```
        n4~dist = INFINITY;

        a1 = "a";
        a2 = "b";
        a3 = "c";
        a4 = "d";


        /* add nodes to g1 */
        g1 = new graph<struct nodedata>();
        g1~+a1;
        g1~+a2;
        g1~+a3;
        g1~+a4;

        a = g1~_a1;
        b = g1~_a2;
        c = g1~_a3;
        d = g1~_a4;

        a.setData(n1);
        g1[1]->(a, b);
        g1[2]->(a, c);

        b.setData(n2);
        g1[1]->(b, a);
        g1[10]->(b, d);

        c.setData(n3);
        g1[2]->(c, a);
        g1[5]->(c, d);

        d.setData(n4);
        g1[10]->(d, b);
        g1[5]->(d, c);

        dijkstra(g1, a);

}
```

fail-assign1.yg

```
int main()
{
  int i;
  bool b;

  i = 42;
  i = 10;
  b = true;
  b = false;
  i = false; /* Fail: assigning a bool to an integer */
```

```
}

Output: Fatal error: exception Failure("illegal assignment int = bool in i
= false")
```

fail-assign2.yg

```
int main()
{
  int i;
  bool b;

  b = 48; /* Fail: assigning an integer to a bool */
}

Output: Fatal error: exception Failure("illegal assignment bool = int in b
= 48")
```

fail-assign3.yg

```
void myvoid()
{
  return;
}

int main()
{
  int i;

  i = myvoid(); /* Fail: assigning a void to an integer */
}

Output: Fatal error: exception Failure("illegal assignment int = void in i
= myvoid()")
```

fail-dead1.yg

```
int main()
{
  int i;

  i = 15;
  return i;
  i = 32; /* Error: code after a return */
}

Output: Fatal error: exception Failure("nothing may follow a return")
```

fail-dead2.yg

```
int main()
```

```
{
  int i;

  {
    i = 15;
    return i;
  }
  i = 32; /* Error: code after a return */
}

Output: Fatal error: exception Failure("nothing may follow a return")
```

fail-expr1.yg

```
int a;
bool b;

void foo(int c, bool d)
{
  int dd;
  bool e;
  a + c;
  c - a;
  a * 3;
  c / 2;
  d + a; /* Error: bool + int */
}

int main()
{
  return 0;
}

Output: Fatal error: exception Failure("illegal binary operator bool + int
in d + a")
```

fail-expr2.yg

```
int a;
bool b;

void foo(int c, bool d)
{
  int d;
  bool e;
  b + a; /* Error: bool + int */
}

int main()
{
  return 0;
}
```

```
Output:

Fatal error: exception Failure("illegal binary operator bool + int in b +
a")
```

fail-for1.yg

```
int main()
{
  int i;
  for ( ; true ; ) {} /* OK: Forever */

  for (i = 0 ; i < 10 ; i = i + 1) {
    if (i == 3) return 42;
  }

  for (j = 0; i < 10 ; i = i + 1) {} /* j undefined */

  return 0;
}

Output: Fatal error: exception Failure("undeclared identifier j")
```

fail-for2.yg

```
int main()
{
  int i;

  for (i = 0; j < 10 ; i = i + 1) {} /* j undefined */

  return 0;
}

Output: Fatal error: exception Failure("undeclared identifier j")
```

fail-for3.yg

```
int main()
{
  int i;

  for (i = 0; i ; i = i + 1) {} /* i is an integer, not Boolean */

  return 0;
}

Output: Fatal error: exception Failure("expected Boolean expression in i")
```

fail-for4.yg

```
int main()
```

```
{
  int i;

  for (i = 0; i < 10 ; i = j + 1) {} /* j undefined */

  return 0;
}

Output: Fatal error: exception Failure("undeclared identifier j")
```

fail-for5.yg

```
int main()
{
  int i;

  for (i = 0; i < 10 ; i = i + 1) {
    foo(); /* Error: no function foo */
  }

  return 0;
}

Output: Fatal error: exception Failure("unrecognized function foo")
```

fail-func1.yg

```
int foo() {}

int bar() {}

int baz() {}

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

int main()
{
  return 0;
}

Output: 14
```

fail-func2.yg

```
int foo(int a, bool b, int c) { }

void bar(int a, bool b, int a) {} /* Error: duplicate formal a in bar */

int main()
{
  return 0;
```

```
}
```

fail-func3.yg

```
int foo(int a, bool b, int c) { }

void bar(int a, void b, int c) {} /* Error: illegal void formal b */

int main()
{
  return 0;
}

Output: Fatal error: exception Failure("illegal void formal b in bar")
```

fail-func4.yg

```
int foo() {}

void bar() {}

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

void baz() {}

int main()
{
  return 0;
}

Output: Fatal error: exception Failure("function print may not be defined")
```

fail-func5.yg

```
int foo() {}

int bar() {
  int a;
  void b; /* Error: illegal void local b */
  bool c;

  return 0;
}

int main()
{
  return 0;
}

Output: Fatal error: exception Failure("illegal void local b in bar")
```

fail-func6.yg

```
void foo(int a, bool b)
{
}

int main()
{
  foo(42, true);
  foo(42); /* Wrong number of arguments */
}

Output: Fatal error: exception Failure("expecting 2 arguments in foo(42)")
```

fail-func7.yg

```
void foo(int a, bool b)
{
}

int main()
{
  foo(42, true);
  foo(42, true, false); /* Wrong number of arguments */
}

Output: Fatal error: exception Failure("expecting 2 arguments in foo(42,
true, false)")
```

fail-func8.yg

```
void foo(int a, bool b)
{
}

void bar()
{
}

int main()
{
  foo(42, true);
  foo(42, bar()); /* int and void, not int and bool */
}

Output: Fatal error: exception Failure("illegal actual argument found void
expected bool in bar()")
```

fail-func9.yg

```
void foo(int a, bool b)
```

```
{
}

int main()
{
  foo(42, true);
  foo(42, 42); /* Fail: int, not bool */
}
```

```
Output: Fatal error: exception Failure("illegal actual argument found int
expected bool in 42")
```

fail-global1.yg

```
int c;
bool b;
void a; /* global variables should not be void */


int main()
{
  return 0;
}
```

```
Output: Fatal error: exception Failure("illegal void global a")
```

fail-global2.yg

```
int b;
bool c;
int a;
int b; /* Duplicate global variable */

int main()
{
  return 0;
}
```

```
Output: Fatal error: exception Failure("duplicate global b")
```

fail-if1.yg

```
int main()
{
  if (true) {}
  if (false) {} else {}
  if (42) {} /* Error: non-bool predicate */
}
```

```
Output: Fatal error: exception Failure("expected Boolean expression in 42")
```

fail-if2.yg

```
int main()
{
  if (true) {
    foo; /* Error: undeclared variable */
  }
}

Output: Fatal error: exception Failure("undeclared identifier foo")
```

fail-if3.yg

```
int main()
{
  if (true) {
    42;
  } else {
    bar; /* Error: undeclared variable */
  }
}

Output: Fatal error: exception Failure("undeclared identifier bar")
```

fail-return1.yg

```
int main()
{
  return true; /* Should return int */
}

Output: Fatal error: exception Failure("return gives bool expected int in
true")
```

fail-return2.yg

```
void foo()
{
  if (true) return 42; /* Should return void */
  else return;
}

int main()
{
  return 42;
}

Output: Fatal error: exception Failure("return gives int expected void in
42")
```

fail-while1.yg

```
int main()
{
  int i;

  while (true) {
    i = i + 1;
  }

  while (42) { /* Should be boolean */
    i = i + 1;
  }

}

Output: Fatal error: exception Failure("expected Boolean expression in 42")
```

fail-while2.yg

```
int main()
{
  int i;

  while (true) {
    i = i + 1;
  }

  while (true) {
    foo(); /* foo undefined */
  }

}

Output: Fatal error: exception Failure("unrecognized function foo")
```

test-add1.yg

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

int main()
{
  int a;
  int b;
  print( add(17, 25) );
  return 0;
}
```

```
Output: 42
```

test-arith1.yg

```
int main()
{
  print(39 + 3);
  return 0;
}

Output: 42
```

test-arith2.yg

```
int main()
{
  print(1 + 2 * 3 + 4);
  return 0;
}

Output: 11
```

test-arith3.yg

```
int foo(int a)
{
  return a;
}

int main()
{
  int a;
  a = 42;
  a = a + 5;
  print(a);
  return 0;
}

Output: 47
```

test-fib.yg

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

int main()
{
  print(fib(0));
```

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

Output:
1
1
2
3
5
8
```

test-fib.yg

```
int main()
{
  int i;
  for (i = 0 ; i < 5 ; i = i + 1) {
    print(i);
  }
  print(42);
  return 0;
}

Output:
0
1
2
3
4
42
```

test-for2.yg

```
int main()
{
  int i;
  i = 0;
  for ( ; i < 5; ) {
    print(i);
    i = i + 1;
  }
  print(42);
  return 0;
}

Output:
```

```
0
1
2
3
4
42
```

test-func1.yg

```
int add(int a, int b)
{
  return a + b;
}

int main()
{
  int a;
  a = add(39, 3);
  print(a);
  return 0;
}

Output: 42
```

test-func2.yg

```
/* Bug noticed by Pin-Chin Huang */

int fun(int x, int y)
{
  return 0;
}

int main()
{
  int i;
  i = 1;

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

  print(i);
  return 0;
}

Output: 2
```

test-func2.yg

```
void printem(int a, int b, int c, int d)
{
  print(a);
```

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

int main()
{
  printem(42,17,192,8);
  return 0;
}

Output:
42
17
192
8
```

test-func4.yg

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

int main()
{
  int d;
  d = add(52, 10);
  print(d);
  return 0;
}

Output: 62
```

test-func5.yg

```
int foo(int a)
{
  return a;
}

int main()
{
  return 0;
}

Output:
```

test-func6.yg

```
void foo() {}

int bar(int a, bool b, int c) { return a + c; }

int main()
{
  print(bar(17, false, 25));
  return 0;
}

Output: 42
```

test-func7.yg

```
int a;

void foo(int c)
{
  a = c + 42;
}

int main()
{
  foo(73);
  print(a);
  return 0;
}

Output: 115
```

test-func8.yg

```
void foo(int a)
{
  print(a + 3);
}

int main()
{
  foo(40);
  return 0;
}

Output: 43
```

test-func8.yg

```
int gcd(int a, int b) {
  while (a != b) {
    if (a > b) a = a - b;
    else b = b - a;
```

```
  }
  return a;
}

int main()
{
  print(gcd(2,14));
  print(gcd(3,15));
  print(gcd(99,121));
  return 0;
}

Output:
2
3
11
```

test-func8.yg

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

int main()
{
  print(gcd(14,21));
  print(gcd(8,36));
  print(gcd(99,121));
  return 0;
}

Output:
7
4
11
```

test-global3.yg

```
int i;
bool b;
int j;

int main()
{
  i = 42;
  j = 10;
  print(i + j);
  return 0;
```

```
}

Output: 52
```

test-global1.yg

```
int a;
int b;

void printa()
{
  print(a);
}

void printb()
{
  print(b);
}

void incab()
{
  a = a + 1;
  b = b + 1;
}

int main()
{
  a = 42;
  b = 21;
  printa();
  printb();
  incab();
  printa();
  printb();
  return 0;
}

Output:
42
21
43
22
```

test-global2.yg

```
bool i;

int main()
{
  int i; /* Should hide the global i */
```

```
  i = 42;
  print(i + i);
  return 0;
}

Output: 84
```

test-hello.yg

```
int main()
{
  print(42);
  print(71);
  print(1);
  return 0;
}

Output:
42
71
1
```

test-if1.yg

```
int main()
{
  if (true) print(42);
  print(17);
  return 0;
}

Output:
42
17
```

test-if2.yg

```
int main()
{
  if (true) print(42); else print(8);
  print(17);
  return 0;
}

Output:
42
17
```

test-if3.yg

```
int main()
```

```
{
  if (false) print(42);
  print(17);
  return 0;
}
Output: 17
```

test-if4.yg

```
int main()
{
  if (false) print(42); else print(8);
  print(17);
  return 0;
}

Output:
8
17
```

test-if5.yg

```
int cond(bool b)
{
  int x;
  if (b)
    x = 42;
  else
    x = 17;
  return x;
}

int main()
{
 print(cond(true));
 print(cond(false));
 return 0;
}

Output:
42
17
```

test-if5.yg

```
void foo(bool i)
{
  int i; /* Should hide the formal i */

  i = 42;
  print(i + i);
```

```
}

int main()
{
  foo(true);
  return 0;
}

Output: 84
```

test-local1.yg

```
void foo(bool i)
{
  int i; /* Should hide the formal i */

  i = 42;
  print(i + i);
}

int main()
{
  foo(true);
  return 0;
}

Output: 84
```

test-local2.yg

```
int foo(int a, bool b)
{
  int c;
  bool d;

  c = a;

  return c + 10;
}

int main() {
 print(foo(37, false));
 return 0;
}

Output: 47
```

test-ops1.yg

```
int main()
{
```

```
  print(1 + 2);
  print(1 - 2);
  print(1 * 2);
  print(100 / 2);
  print(99);
  printb(1 == 2);
  printb(1 == 1);
  print(99);
  printb(1 != 2);
  printb(1 != 1);
  print(99);
  printb(1 < 2);
  printb(2 < 1);
  print(99);
  printb(1 <= 2);
  printb(1 <= 1);
  printb(2 <= 1);
  print(99);
  printb(1 > 2);
  printb(2 > 1);
  print(99);
  printb(1 >= 2);
  printb(1 >= 1);
  printb(2 >= 1);
  return 0;
}

Output:
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-ops2.yg

```
int main()
{
  printb(true);
  printb(false);
  printb(true && true);
  printb(true && false);
  printb(false && true);
  printb(false && false);
  printb(true || true);
  printb(true || false);
  printb(false || true);
  printb(false || false);
  printb(!false);
  printb(!true);
  print(-10);
  print(--42);
}

Output:
1
0
1
0
0
0
1
1
1
0
1
0
-10
```

test-printbig.yg

```
/*
 * Test for linking external C functions to LLVM-generated code
 *
 * printbig is defined as an external function, much like printf
 * The C compiler generates printbig.o
 * The LLVM compiler, llc, translates the .ll to an assembly .s file
 * The C compiler assembles the .s file and links the .o file to generate
 * an executable
 */

int main()
{
  printbig(72); /* H */
  printbig(69); /* E */
```

```
  printbig(76); /* L */
  printbig(76); /* L */
  printbig(79); /* O */
  printbig(32); /*   */
  printbig(87); /* W */
  printbig(79); /* O */
  printbig(82); /* R */
  printbig(76); /* L */
  printbig(68); /* D */
  return 0;
}
```

Output:

```
  XXXXXXXXXXXXXX
  XXXXXXXXXXXXXX
  XX    XX    XX
  XX    XX    XX
  XX    XX    XX
  XX          XX


  XXXXXXXXXXXXXX
  XXXXXXXXXXXXXX
  XX
  XX
  XX
  XX


  XXXXXXXXXXXXXX
  XXXXXXXXXXXXXX
  XX
  XX
  XX
  XX

    XXXXXXXXXX
  XXXXXXXXXXXXXX
  XX          XX
  XX          XX
  XX          XX
  XXXXXXXXXXXXXX
    XXXXXXXXXX
```

```
     XXXXXXXXXX
  XXXXXXXXXXXXXX
    XXXXXX
      XXXXXX
    XXXXXX
  XXXXXXXXXXXXXX
     XXXXXXXXXX

    XXXXXXXXXX
  XXXXXXXXXXXXXX
  XX          XX
  XX          XX
  XX          XX
  XXXXXXXXXXXXXX
    XXXXXXXXXX

  XXXXXXXXXXXXXX
  XXXXXXXXXXXXXX
     XX      XX
    XXXX     XX
  XXXXXXXX   XX
  XXXX  XXXXXXXX
  XX    XXXXXX


  XXXXXXXXXXXXXX
  XXXXXXXXXXXXXX
  XX
  XX
  XX
  XX

  XXXXXXXXXXXXXX
  XXXXXXXXXXXXXX
  XX          XX
  XX          XX
  XXXX      XXXX
    XXXXXXXXXX
      XXXXXX
```

test-printbig.yg

```
int main()
{
  float f;
  f = 42.0;
  printfloat(f);
  return 0;
}
```

```
42.100000
```

test-queueadd.yg

```
int main() {

  int a;
  int b;
  int c;
  Queue<int> q;
  a = 3;
  c = 5;
  q = new Queue<int>();
  q.qadd(a);
  q.qremove();
  q.qadd(c);
  b = q.qfront();
  print(b);

  return 0;
}

Output: 5
```

test-queuedecl.yg

```
int main() {

  int a;
  Queue<int> q;
  q = new Queue<int>(a);
  return 0;
}

Output:
```

test-stringdecl.yg

```
int main () {
       string s;
       s = "hello";
       return 0;
}

Output:
```

test-stringprint.yg

```
int main () {
       string s;
       s = "hello world";
```

```
      prints(s);
      return 0;
}

Output: hello world
```

test-var1.yg

```
int main()
{
  int a;
  a = 42;
  print(a);
  return 0;
}

Output: 42
```

test-var2.yg

```
int a;

void foo(int c)
{
  a = c + 42;
}

int main()
{
  foo(73);
  print(a);
  return 0;
}

Output: 115
```

test-while1.yg

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

Output:
```

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

test-while2.yg

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

int main()
{
  print(foo(7));
  return 0;
}
Output: 14
```