Circline -- An Easy Graph Language

Haikuo Liu
Jia zhang
Qing Lan
Zehao Song

Language Summary

Basic: Integer, Floating Point 64 bit, Boolean, String, Null

Data Structure: List, Dict, Node, Graph

Operations: Arithmetic Operation, Logic Operation, Conditional

Operation, Graph Operation

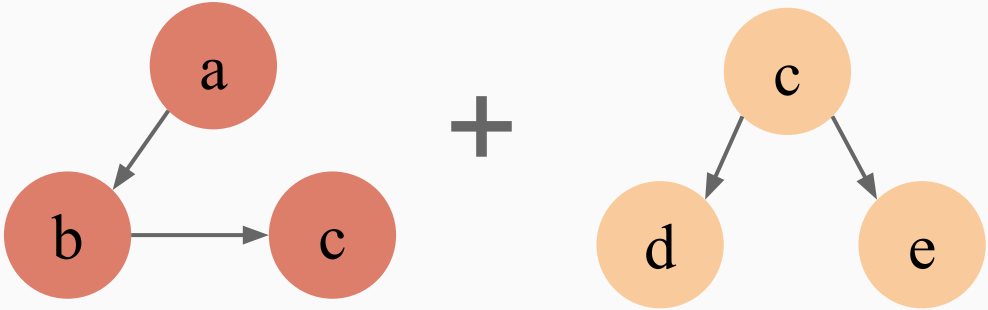
Language Feature

- Native Support on Node, Edge and Graph Definition & Operation
- Function and variables declared everywhere, Support nested function
- Support List, Hashmap basic Data structure

Circle + Line = Circline

Node & Graph - Merge Graph

```
node a = node(“a”);

graph g1 = a -> b -> c;

graph g2 = c -> [d, e];

graph gh = g1 + g2;
```



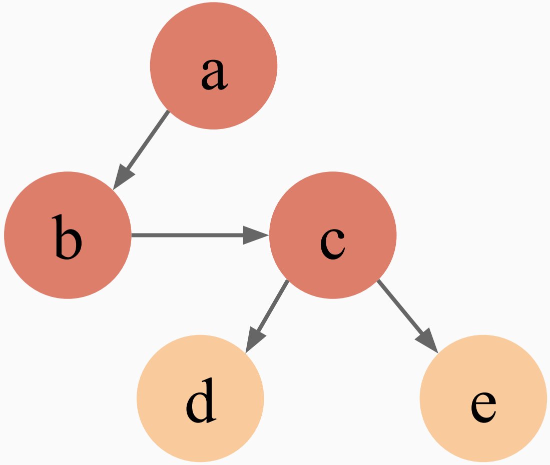
a -> b -> c

a -> (b -> c)

c -> [d, e]

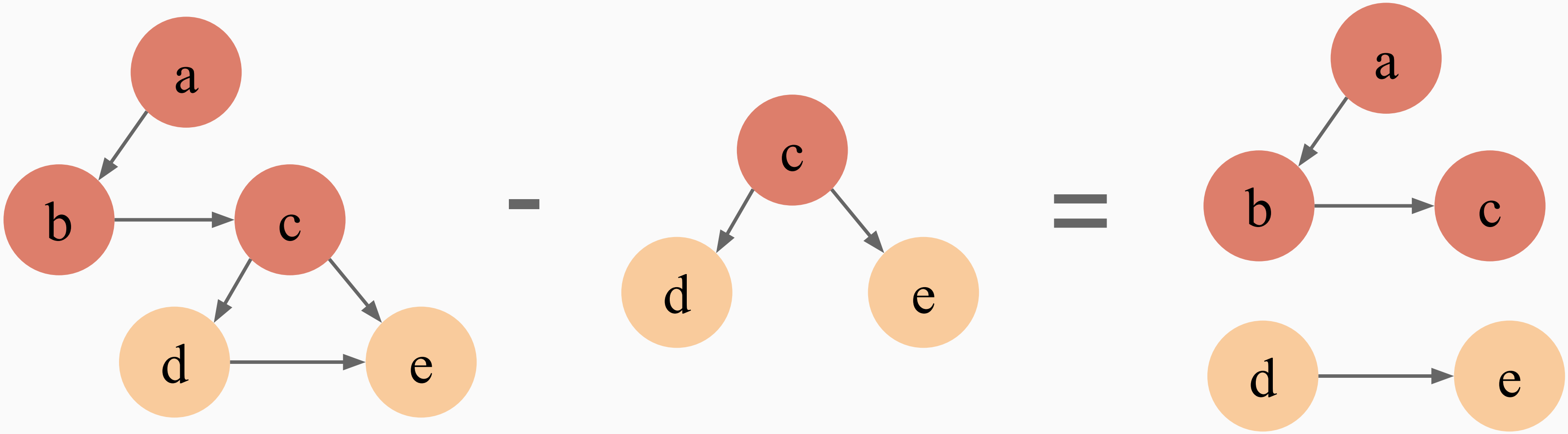
Node & Graph - Merge Graph

```
node a = node(“a”);

graph g1 = a -> b -> c;

graph g2 = c -> [d, e];

graph gh = g1 + g2;
```



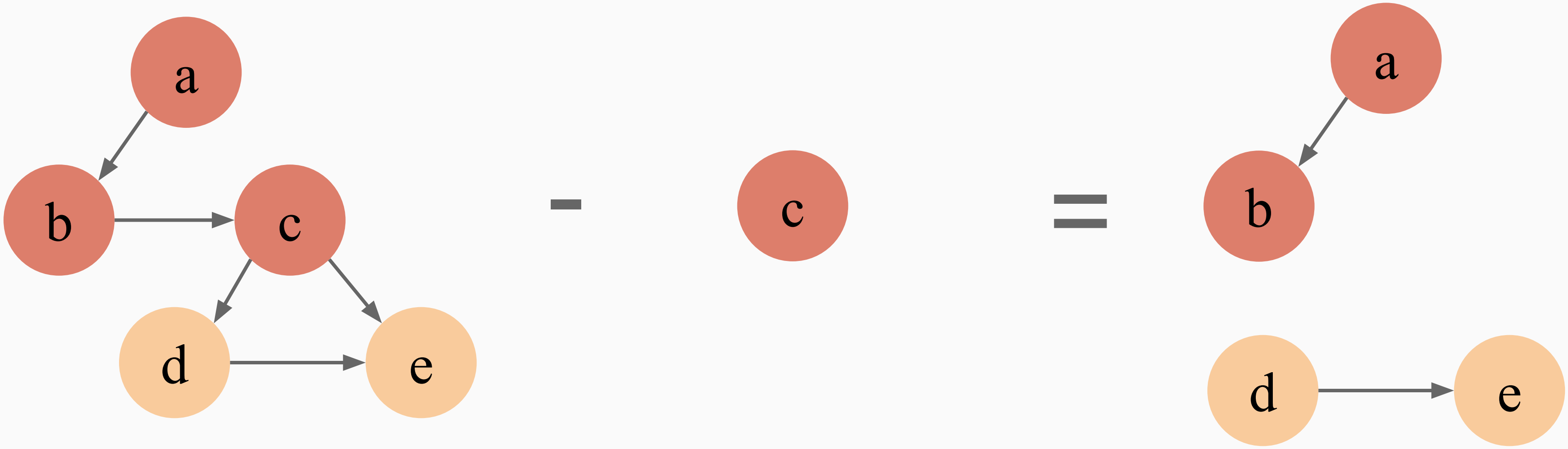
```
a -> b -> c -> [d, e]
```

Node & Graph - Graph Subtraction



```
  a -> b -> c
-> [d -> e, e]     -     c -> [d, e]     =     list<graph>
```

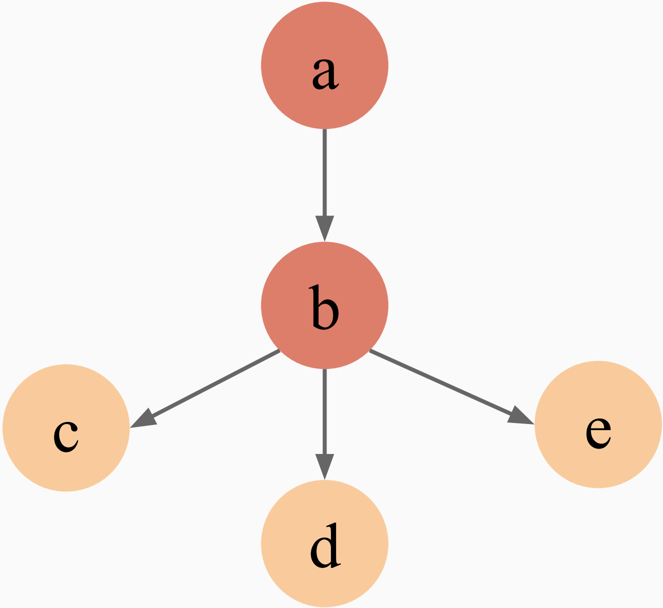
Node & Graph - Node Removal



```
   a -> b -> c         -      c      =    list<graph>
-> [d -> e, e]
```

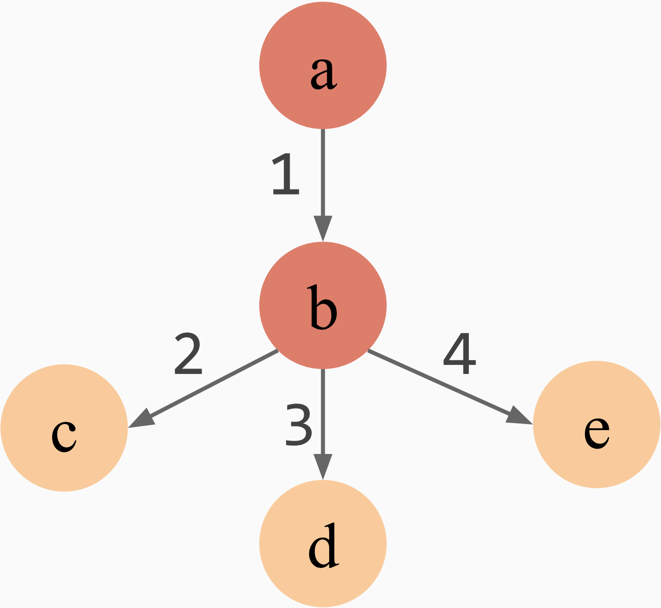
Node & Graph - Neighbors

```
graph gh = a -> b -> [c, d, e]


 gh @ a => [ b ]

 gh @ b => [c, d, e]

 gh @ c => []
```



Node & Graph - Edge Value

```
graph gh = a -> 1&b -> [2&c, 3&d, 4&e]
graph gh = a -> 1&b -> [2,3,4]& [c,d,e]

    gh @ (a, b) => 1

    gh @ (b, a) => null

    gh @ (b, d) => 3
```



List

```
list<int> li = [ 1, 2, 3];
```

Auto Conversion

```
list<float> lf = [1, 1.2, 3];

list<graph> lg = [a, a -> b];
```

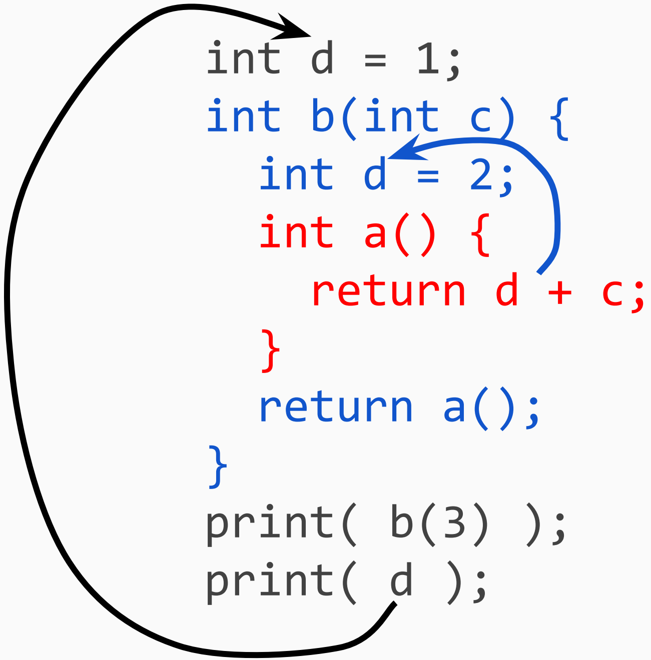
- **Array**
  - get()
  - set()
- **Queue**
  - add()
  - remove()
- **Stack**
  - push()
  - pop()

Dict

```
node a = node("a");
node b = node("b");

dict<node> set = { a: a };
set.has(a) => true
set.get(b) => false
set.get(a) => 1
```

- **Map**
  - put()
  - get()
- **Set**
  - has()
  - keys()

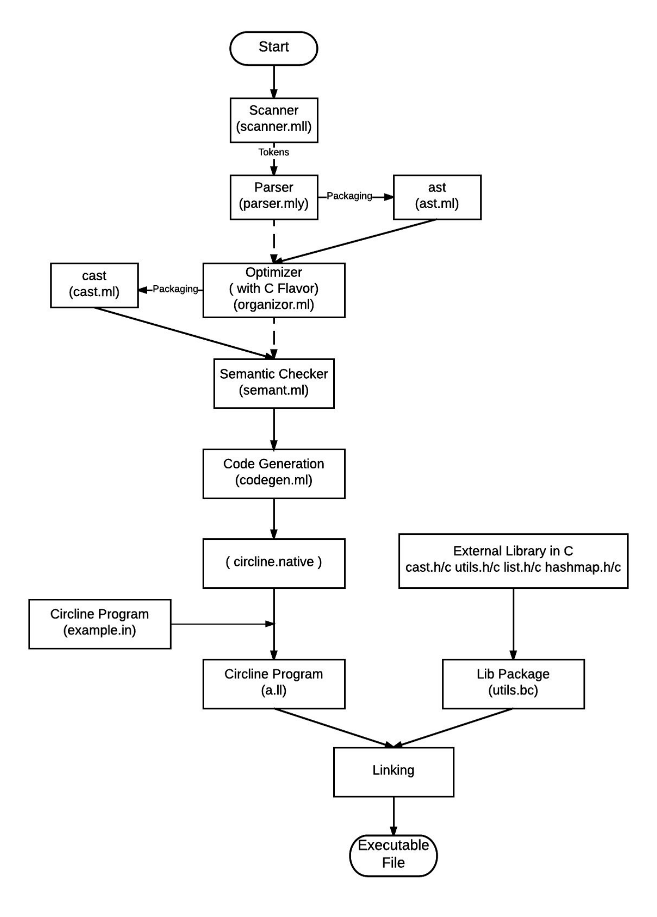
Nested Functions



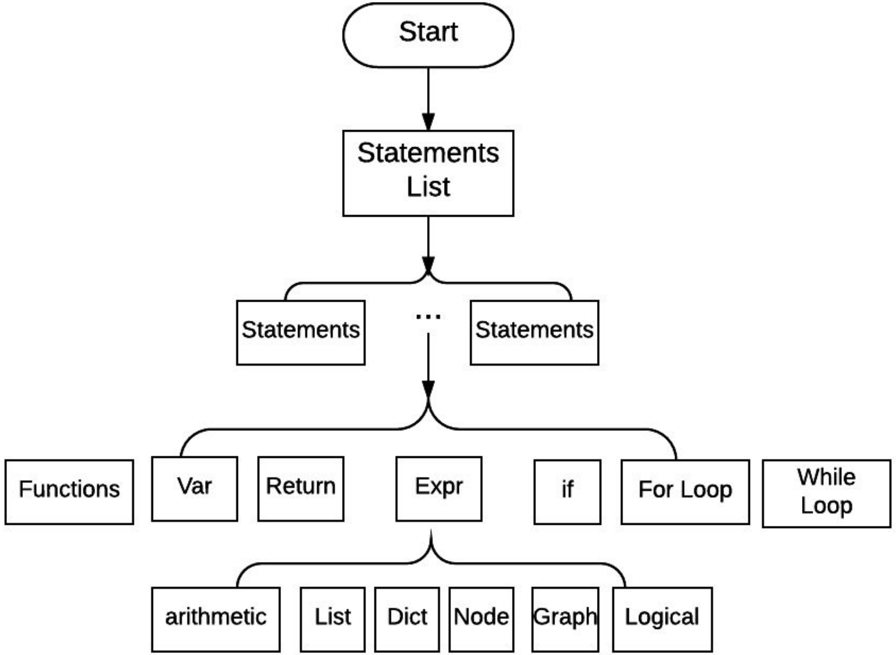
```
int d = 1;
int b(int c) {
   int d = 2;
   int a() {
      return d + c;
   }
   return a();
}
print( b(3) );         /* Output 5 */
print( d );            /* Output 1 */
```

- Access Outer Variables
- Scoping - Static

System Architechture

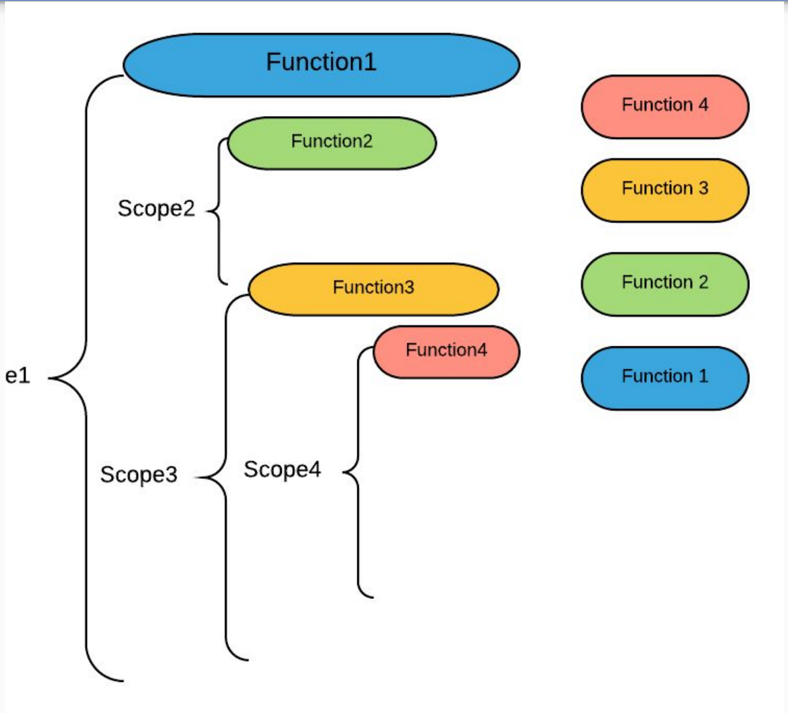


Scanner/Parser



Organizer

- A bridge between Circline and C
- Format the functions and variables



Semantic Check

The cast returned by Organizer is a list of function objects.

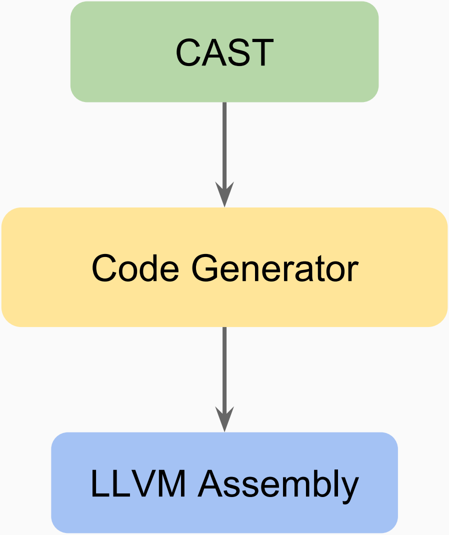
cast: [func1, func2, …, funcn]

Loop through all function objects and check each function objects.

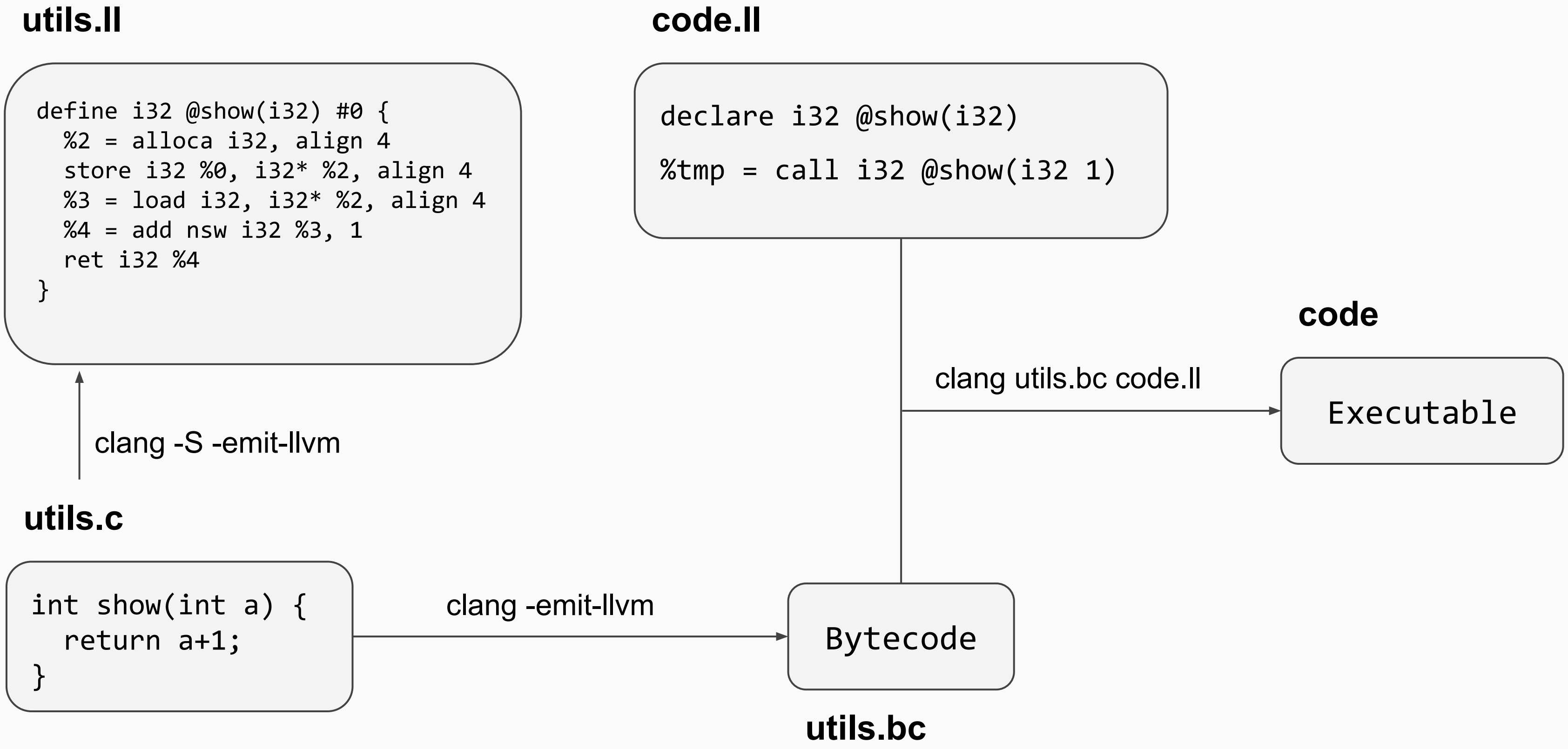
For nest scope situation, we try to search in parent scope if the variable is not found in current scope.

Code Generation - CAST to LLVM Assembly

```
declare external functions (C Libraries)
for function in program:
    declare all variables in function
    for statement in function:
        for expression in statement:
            codegen( expression )
```



Code Generator - C Library



utils.ll

```
define i32 @show(i32) #0 {
  %2 = alloca i32, align 4
  store i32 %0, i32* %2, align 4
  %3 = load i32, i32* %2, align 4
  %4 = add nsw i32 %3, 1
  ret i32 %4
}
```

clang -S -emit-llvm

utils.c

```
int show(int a) {
  return a+1;
}
```

clang -emit-llvm

Bytecode

utils.bc

code.ll

```
declare i32 @show(i32)

%tmp = call i32 @show(i32 1)
```

clang utils.bc code.ll

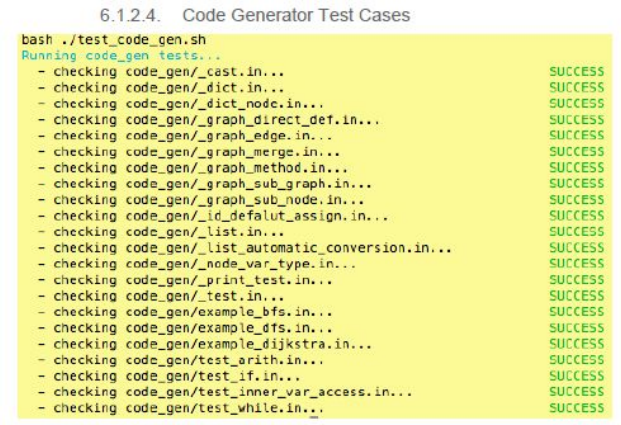
code

Executable

Automated Build and Test -- Save Time!

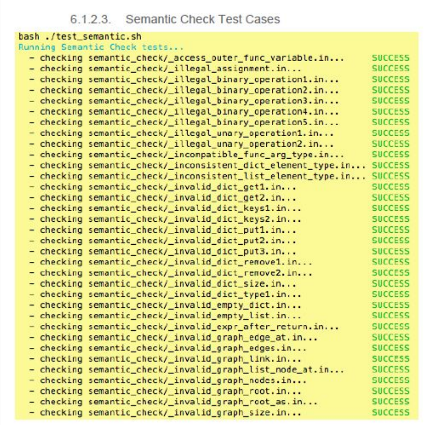
Makefile: make all/test (Find target and build build build)

Travis-CI Online Code check



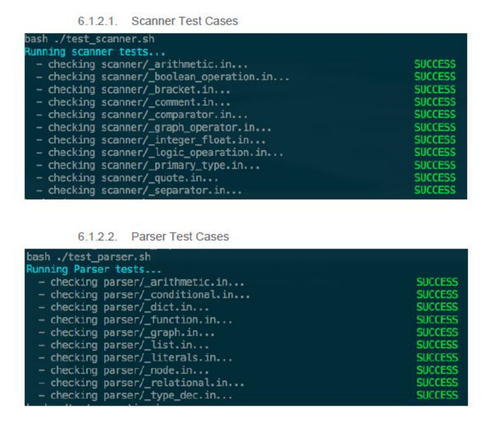
6.1.2.4. Code Generator Test Cases

```
bash ./test_code_gen.sh
Running code_gen tests...
  - checking code_gen/_cast.in...                              SUCCESS
  - checking code_gen/_dict.in...                              SUCCESS
  - checking code_gen/_dict_node.in...                         SUCCESS
  - checking code_gen/_graph_direct_def.in...                  SUCCESS
  - checking code_gen/_graph_edge.in...                        SUCCESS
  - checking code_gen/_graph_merge.in...                       SUCCESS
  - checking code_gen/_graph_method.in...                      SUCCESS
  - checking code_gen/_graph_sub_graph.in...                   SUCCESS
  - checking code_gen/_graph_sub_node.in...                    SUCCESS
  - checking code_gen/_id_defalut_assign.in...                 SUCCESS
  - checking code_gen/_list.in...                              SUCCESS
  - checking code_gen/_list_automatic_conversion.in...         SUCCESS
  - checking code_gen/_node_var_type.in...                     SUCCESS
  - checking code_gen/_print_test.in...                        SUCCESS
  - checking code_gen/_test.in...                              SUCCESS
  - checking code_gen/example_bfs.in...                        SUCCESS
  - checking code_gen/example_dfs.in...                        SUCCESS
  - checking code_gen/example_dijkstra.in...                   SUCCESS
  - checking code_gen/test_arith.in...                         SUCCESS
  - checking code_gen/test_if.in...                            SUCCESS
  - checking code_gen/test_inner_var_access.in...              SUCCESS
  - checking code_gen/test_while.in...                         SUCCESS
```



6.1.2.3. Semantic Check Test Cases

```
bash ./test_semantic.sh
Running Semantic Check tests...
  - checking semantic_check/_access_outer_func_variable.in...     SUCCESS
  - checking semantic_check/_illegal_assignment.in...             SUCCESS
  - checking semantic_check/_illegal_binary_operation1.in...      SUCCESS
  - checking semantic_check/_illegal_binary_operation2.in...      SUCCESS
  - checking semantic_check/_illegal_binary_operation3.in...      SUCCESS
  - checking semantic_check/_illegal_binary_operation4.in...      SUCCESS
  - checking semantic_check/_illegal_binary_operation5.in...      SUCCESS
  - checking semantic_check/_illegal_unary_operation1.in...       SUCCESS
  - checking semantic_check/_illegal_unary_operation2.in...       SUCCESS
  - checking semantic_check/_incompatible_func_arg_type.in...     SUCCESS
  - checking semantic_check/_inconsistent_dict_element_type.in... SUCCESS
  - checking semantic_check/_inconsistent_list_element_type.in... SUCCESS
  - checking semantic_check/_invalid_dict_get1.in...              SUCCESS
  - checking semantic_check/_invalid_dict_get2.in...              SUCCESS
  - checking semantic_check/_invalid_dict_keys1.in...             SUCCESS
  - checking semantic_check/_invalid_dict_keys2.in...             SUCCESS
  - checking semantic_check/_invalid_dict_put1.in...              SUCCESS
  - checking semantic_check/_invalid_dict_put2.in...              SUCCESS
  - checking semantic_check/_invalid_dict_put3.in...              SUCCESS
  - checking semantic_check/_invalid_dict_remove1.in...           SUCCESS
  - checking semantic_check/_invalid_dict_remove2.in...           SUCCESS
  - checking semantic_check/_invalid_dict_size.in...              SUCCESS
  - checking semantic_check/_invalid_dict_type1.in...             SUCCESS
  - checking semantic_check/_invalid_empty_dict.in...             SUCCESS
  - checking semantic_check/_invalid_empty_list.in...             SUCCESS
  - checking semantic_check/_invalid_expr_after_return.in...      SUCCESS
  - checking semantic_check/_invalid_graph_edge_at.in...          SUCCESS
  - checking semantic_check/_invalid_graph_edges.in...            SUCCESS
  - checking semantic_check/_invalid_graph_link.in...             SUCCESS
  - checking semantic_check/_invalid_graph_list_node_at.in...     SUCCESS
  - checking semantic_check/_invalid_graph_nodes.in...            SUCCESS
  - checking semantic_check/_invalid_graph_root.in...             SUCCESS
  - checking semantic_check/_invalid_graph_root_as.in...          SUCCESS
  - checking semantic_check/_invalid_graph_size.in...             SUCCESS
```



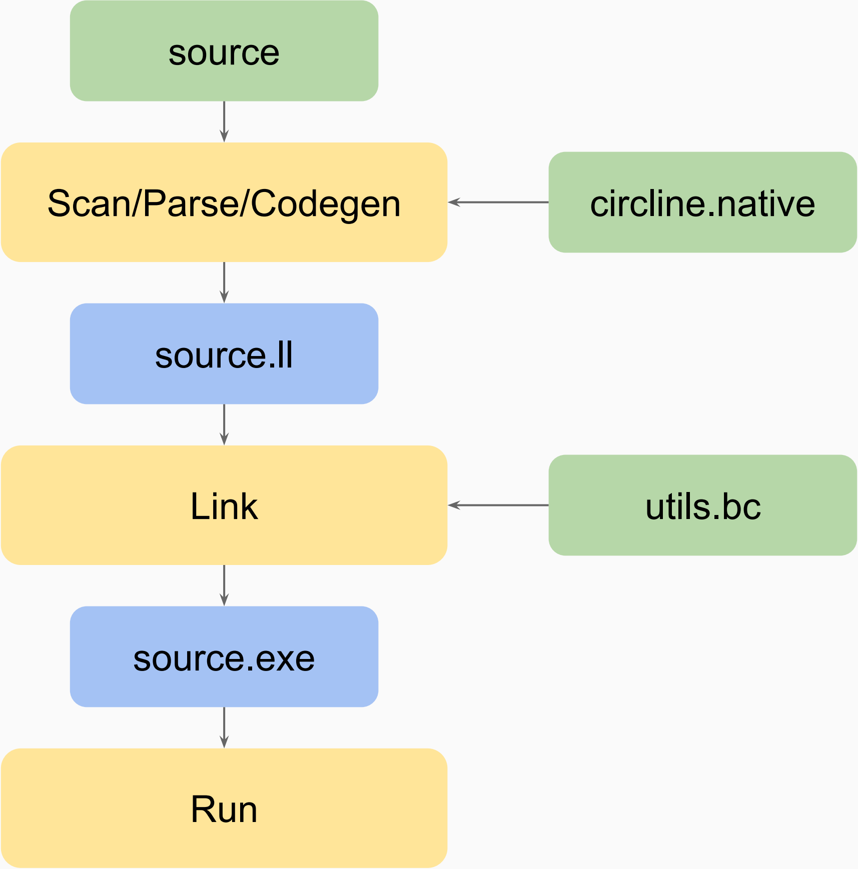
6.1.2.1. Scanner Test Cases

```
bash ./test_scanner.sh
Running scanner tests...
  - checking scanner/_arithmetic.in...                         SUCCESS
  - checking scanner/_boolean_operation.in...                  SUCCESS
  - checking scanner/_bracket.in...                            SUCCESS
  - checking scanner/_comment.in...                            SUCCESS
  - checking scanner/_comparator.in...                         SUCCESS
  - checking scanner/_graph_operator.in...                     SUCCESS
  - checking scanner/_integer_float.in...                      SUCCESS
  - checking scanner/_logic_opearation.in...                   SUCCESS
  - checking scanner/_primary_type.in...                       SUCCESS
  - checking scanner/_quote.in...                              SUCCESS
  - checking scanner/_separator.in...                          SUCCESS
```

6.1.2.2. Parser Test Cases

```
bash ./test_parser.sh
Running Parser tests...
  - checking parser/_arithmetic.in...                          SUCCESS
  - checking parser/_conditional.in...                         SUCCESS
  - checking parser/_dict.in...                                SUCCESS
  - checking parser/_function.in...                            SUCCESS
  - checking parser/_graph.in...                               SUCCESS
  - checking parser/_list.in...                                SUCCESS
  - checking parser/_literals.in...                            SUCCESS
  - checking parser/_node.in...                                SUCCESS
  - checking parser/_relational.in...                          SUCCESS
  - checking parser/_type_dec.in...                            SUCCESS
```

Compile & Run

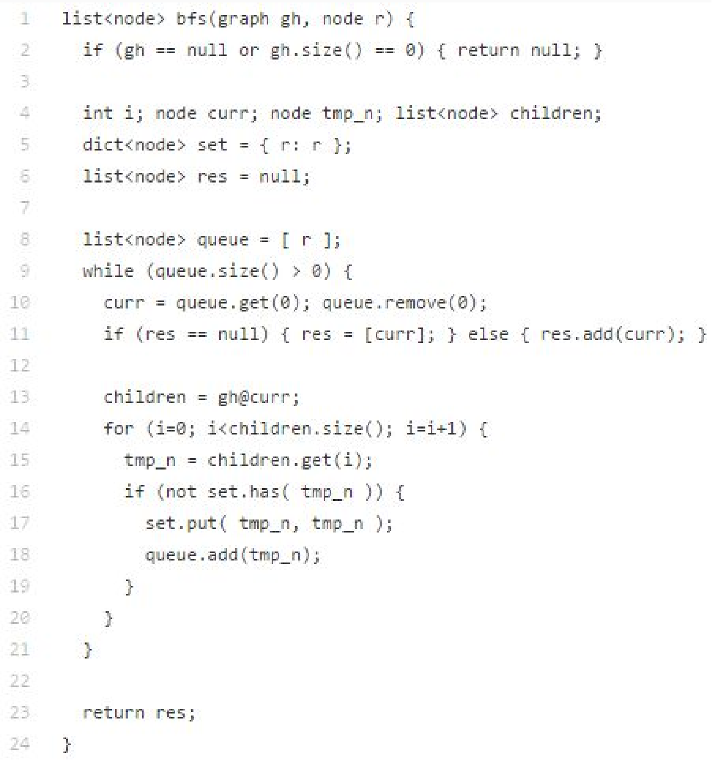


sh circline.sh <code>

Let’s try to run it!

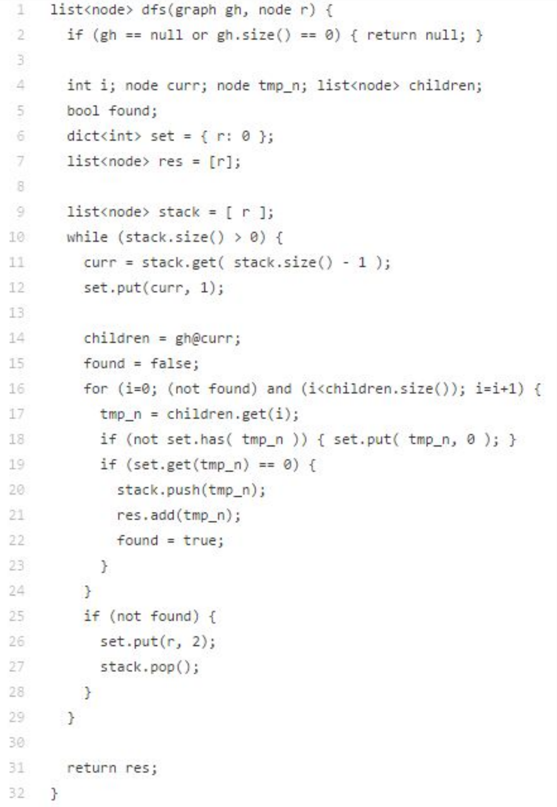
Case Study -- BFS & DFS

BFS Code



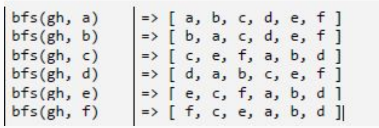
```
list<node> bfs(graph gh, node r) {
  if (gh == null or gh.size() == 0) { return null; }

  int i; node curr; node tmp_n; list<node> children;
  dict<node> set = { r: r };
  list<node> res = null;

  list<node> queue = [ r ];
  while (queue.size() > 0) {
    curr = queue.get(0); queue.remove(0);
    if (res == null) { res = [curr]; } else { res.add(curr); }

    children = gh@curr;
    for (i=0; i<children.size(); i=i+1) {
      tmp_n = children.get(i);
      if (not set.has( tmp_n )) {
        set.put( tmp_n, tmp_n );
        queue.add(tmp_n);
      }
    }
  }

  return res;
}
```

DFS Code



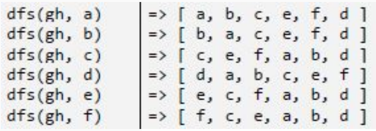
```
list<node> dfs(graph gh, node r) {
  if (gh == null or gh.size() == 0) { return null; }

  int i; node curr; node tmp_n; list<node> children;
  bool found;
  dict<int> set = { r: 0 };
  list<node> res = [r];

  list<node> stack = [ r ];
  while (stack.size() > 0) {
    curr = stack.get( stack.size() - 1 );
    set.put(curr, 1);

    children = gh@curr;
    found = false;
    for (i=0; (not found) and (i<children.size()); i=i+1) {
      tmp_n = children.get(i);
      if (not set.has( tmp_n )) { set.put( tmp_n, 0 ); }
      if (set.get(tmp_n) == 0) {
        stack.push(tmp_n);
        res.add(tmp_n);
        found = true;
      }
    }
    if (not found) {
      set.put(r, 2);
      stack.pop();
    }
  }

  return res;
}
```

BFS Printout

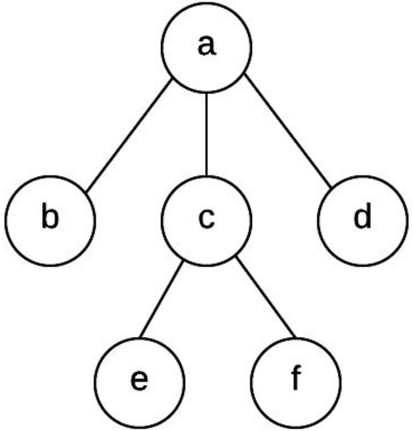


```
bfs(gh, a)    => [ a, b, c, d, e, f ]
bfs(gh, b)    => [ b, a, c, d, e, f ]
bfs(gh, c)    => [ c, e, f, a, b, d ]
bfs(gh, d)    => [ d, a, b, c, e, f ]
bfs(gh, e)    => [ e, c, f, a, b, d ]
bfs(gh, f)    => [ f, c, e, a, b, d ]
```

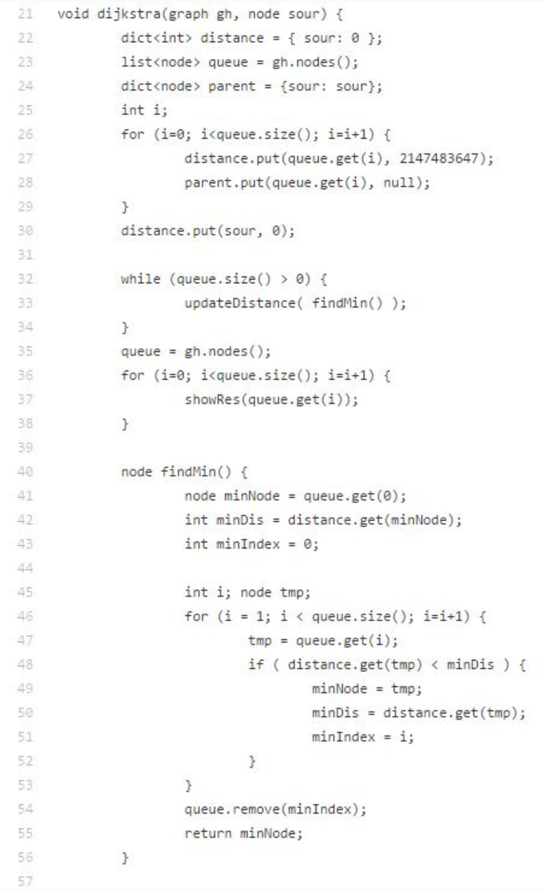
DFS Printout



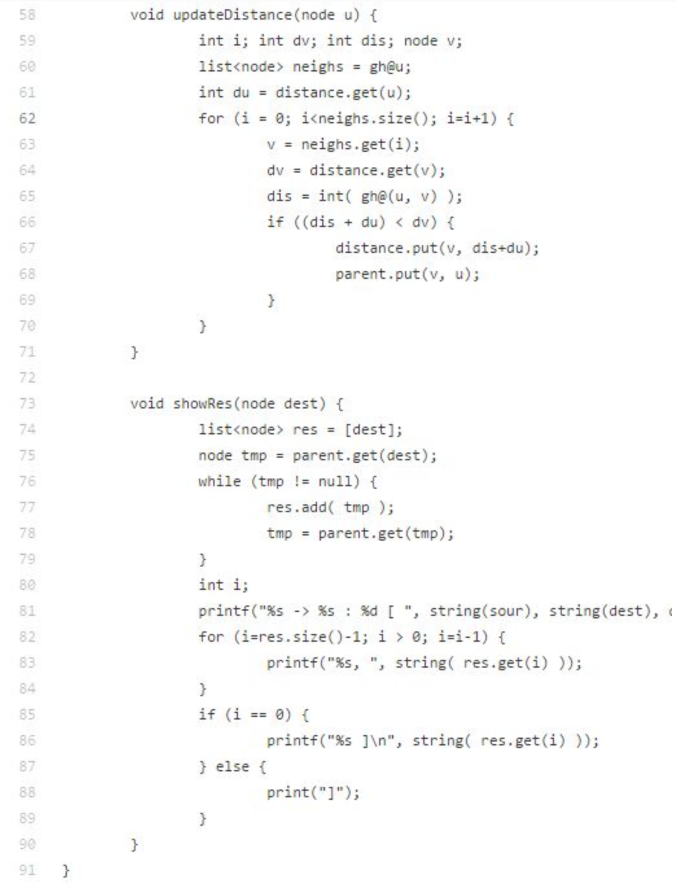
```
dfs(gh, a)    => [ a, b, c, e, f, d ]
dfs(gh, b)    => [ b, a, c, e, f, d ]
dfs(gh, c)    => [ c, e, f, a, b, d ]
dfs(gh, d)    => [ d, a, b, c, e, f ]
dfs(gh, e)    => [ e, c, f, a, b, d ]
dfs(gh, f)    => [ f, c, e, a, b, d ]
```



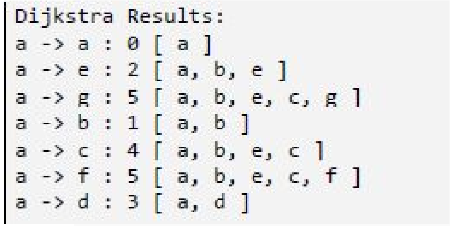
Case Study -- Dijkstra Algorithm



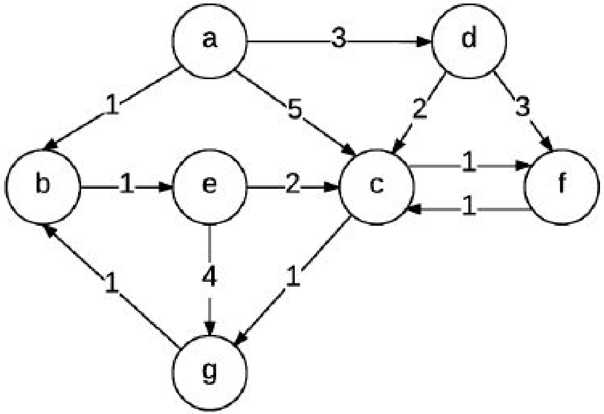
```
void dijkstra(graph gh, node sour) {
    dict<int> distance = { sour: 0 };
    list<node> queue = gh.nodes();
    dict<node> parent = {sour: sour};
    int i;
    for (i=0; i<queue.size(); i=i+1) {
        distance.put(queue.get(i), 2147483647);
        parent.put(queue.get(i), null);
    }
    distance.put(sour, 0);

    while (queue.size() > 0) {
        updateDistance( findMin() );
    }
    queue = gh.nodes();
    for (i=0; i<queue.size(); i=i+1) {
        showRes(queue.get(i));
    }

    node findMin() {
        node minNode = queue.get(0);
        int minDis = distance.get(minNode);
        int minIndex = 0;

        int i; node tmp;
        for (i = 1; i < queue.size(); i=i+1) {
            tmp = queue.get(i);
            if ( distance.get(tmp) < minDis ) {
                minNode = tmp;
                minDis = distance.get(tmp);
                minIndex = i;
            }
        }
        queue.remove(minIndex);
        return minNode;
    }
```



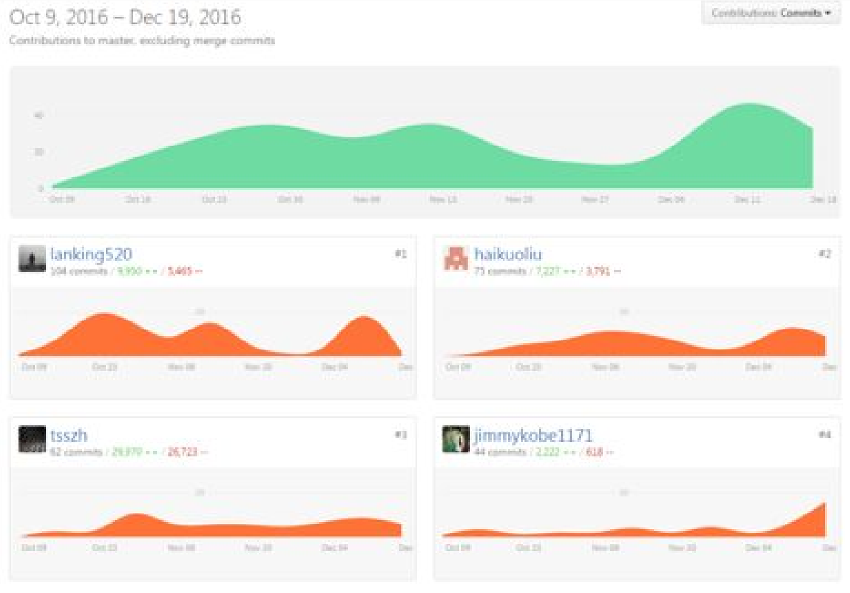
```
    void updateDistance(node u) {
        int i; int dv; int dis; node v;
        list<node> neighs = gh@u;
        int du = distance.get(u);
        for (i = 0; i<neighs.size(); i=i+1) {
            v = neighs.get(i);
            dv = distance.get(v);
            dis = int( gh@(u, v) );
            if ((dis + du) < dv) {
                distance.put(v, dis+du);
                parent.put(v, u);
            }
        }
    }

    void showRes(node dest) {
        list<node> res = [dest];
        node tmp = parent.get(dest);
        while (tmp != null) {
            res.add( tmp );
            tmp = parent.get(tmp);
        }
        int i;
        printf("%s -> %s : %d [ ", string(sour), string(dest), (
        for (i=res.size()-1; i > 0; i=i-1) {
            printf("%s, ", string( res.get(i) ));
        }
        if (i == 0) {
            printf("%s ]\n", string( res.get(i) ));
        } else {
            print("]");
        }
    }
}
```



```
Dijkstra Results:
a -> a : 0 [ a ]
a -> e : 2 [ a, b, e ]
a -> g : 5 [ a, b, e, c, g ]
a -> b : 1 [ a, b ]
a -> c : 4 [ a, b, e, c ]
a -> f : 5 [ a, b, e, c, f ]
a -> d : 3 [ a, d ]
```



Project Timeline & Contribution



Oct 9, 2016 – Dec 19, 2016

Contributions to master, excluding merge commits

lanking520 #1

104 commits / 9,950 ++ / 5,465 --

haikuoliu #2

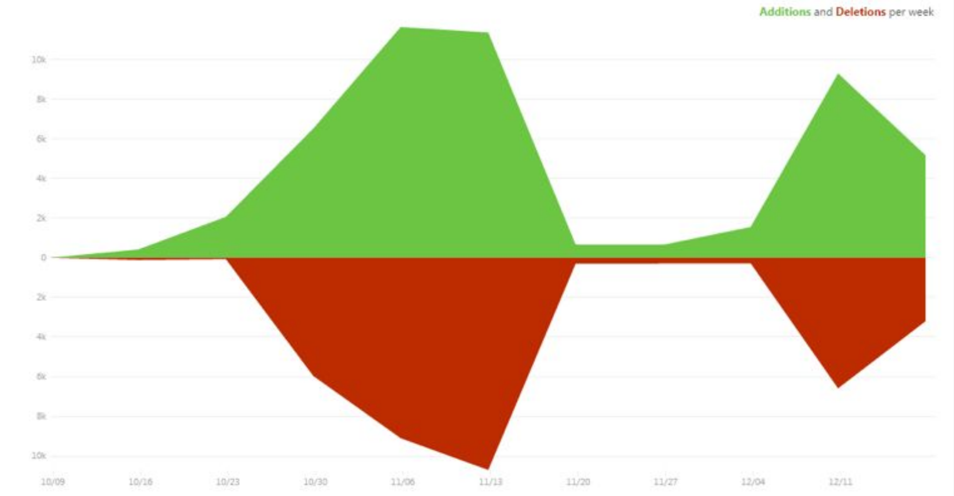
75 commits / 7,227 ++ / 3,791 --

tsszh #3

62 commits / 29,979 ++ / 26,723 --

jimmykobe1171 #4

44 commits / 2,222 ++ / 618 --



With Special Thanks to Alexandra, our TA

who continuously support our project