YAGL
Yet Another Graphing Language
Adam Carpentieri (ac4409)
James Mastran (jam2454)
Jack Hurley (jth2165)
Shvetank Prakash (sp3816)

Final Presentation
Meet The Team

Adam Carpentieri  
Manager

James Mastran  
Language Guru

Shvetank Prakash  
System Architect

Jack Hurley  
Tester
Agenda

1. Target Audience & Motivation
2. YAGL In One Slide
3. Compiler Architecture
4. Cool YAGL Components
5. YAGL Standard Library
6. Built-in Functions (C)
7. Who Did What?
8. Demo
9. Q&A
About YAGL: Target Audience & Motivation

- Pervasiveness of graphs in CS ⇒ great candidate
  - fundamental in data structures & algorithms
- Aims to make implementing graphs & algorithms much simpler
- Ubiquitous with numerous applications:
  - social media connections
  - roads that connect cities
  - flights between cities
  - many other mathematical & logical problems
- Statically & Strongly typed, imperative language
- C-like syntax but adopted other languages features we appreciated
YAGL in One Slide

- No main()
- Import standard library for `print_graph_lib`
- Declaring and initializing graphs and nodes (more later)
- Adding nodes and edges to graphs
- Scoping

```java
import stdgraph.ygl

Graph example;
Node n(gen_name("1"));
Node n2(gen_name("2"));

example: + n + n2, n -> n2 -> n;

print_graph_lib(example);

String gen_name(String n) {
    String hidden = "ODE";
    String total = "";
    {
        String hidden = "N";
        total = total + hidden;
    }
    return total + hidden + n;
}
```
Compiler Architecture

1. Preprocessor
   Input .ygl

2. Scanner
   Tokens

3. Parser
   AST

4. Semantic Checker
   Graph Functions .c
   Linker (C Compiler)

5. Assembly Code
   LLVM IR

6. Code Generation
   SAST

7. executable
Cool YAGL Features
No main()

- Each file has an implicit main function
  - Entry point

- Implemented via lifting all “orphaned” statements
  - Statements not within a function

```cpp
printString("Hello World!");
```
Generic printing capabilities

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

```haskell
in List.fold_left add_bind StringMap.empty [("print", Void);

<table>
<thead>
<tr>
<th>SCall(&quot;print&quot;, [x]) -&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>match x with</td>
</tr>
<tr>
<td>(Node, _) -&gt; (L.build_call print_node_func [ expr builder s_table x ] &quot;print_node&quot; builder)</td>
</tr>
<tr>
<td>(Graph, _) -&gt; L.build_call print_graph_func [ expr builder s_table x ] &quot;print_graph&quot; builder</td>
</tr>
<tr>
<td>(Int, _)</td>
</tr>
<tr>
<td>(Char, _) -&gt; L.build_call printf_func [ char_format_str ; expr builder s_table x ] &quot;printf&quot; builder</td>
</tr>
<tr>
<td>(Float, _) -&gt; L.build_call printf_func [ float_format_str ; expr builder s_table x ] &quot;printf&quot; builder</td>
</tr>
<tr>
<td>(String, _) -&gt; L.build_call printf_func [ string_format_str ; expr builder s_table x ] &quot;printf&quot; builder</td>
</tr>
<tr>
<td>_ -&gt; raise (Failure(&quot;Not implemented print type.&quot;))</td>
</tr>
</tbody>
</table>
```
Arrays

- Different syntax than C
  - int[10] foo vs int foo[10]

- Flexible in ways it can be used and accessed
  - Arrays of all types and any [expr] inside

- LLVM `getelementptr understanding key`
Scoping

- Each block has his own scope
  - C-like scoping rules

- Variables not just declared at top

- Implemented via symbol tables
  - Semantic Checker & Codegen pass around a list of symbol tables

```c
/* Every single variable is named foo */
String foo = "wins";
{
    String foo = "always";
    {
        bool foo = true;
        {
            if(foo) {
                int foo = 42;
                printInt(foo);
            }
        }
    }
}
printString(foo);
printString(foo);
```
Preprocessing / Importing

- **import** keyword
  - Python inspiration
  - Acts similar to C’s preprocessing directives

- File imported is “pasted” to provide access to all functions and vars

- Done prior to feeding to scanner

```python
import stdalgo.ygl
import stdgraph.ygl

/* Social Media Application */
Graph fb; /* Facebook */
Graph tw; /* Twitter */

/* Find friends of friends */
dfs(fb, james, 2); /* S
print(fb);
```
Graphs with Nodes & Edges

- Allocates room for empty Graph on the heap
  - Graphs dynamically grow to hold “infinite” nodes

- Allocates and initializes a node with given name

- Adds Node to the team Graph
  - Nodes can be placed in multiple graphs

- Recursively adds Edges to graph with default weight of 1

```java
Graph team;
Node adam("Adam");
Node james("James");
Node tank("Tank");
Node jack("Jack");
team + adam;
team + james;
team + tank;
team + jack;
team: adam -> james -> tank -> jack -> adam;
printGraph(team);
```

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

Edges:
[(0) : Adam] ---(1)--> [(1) : James]
[(1) : James] ---(1)--> [(2) : Tank]
[(2) : Tank] ---(1)--> [(3) : Jack]
[(3) : Jack] ---(1)--> [(0) : Adam]
More Complex Augmentations

- Complicated example to show variety of graph operation in single LOC
  - Add Node +
  - Add Edge -> and <-
  - Add Bidirectional Edge <->

```javascript
1  Graph demo;
2  Node portland("Portland");
3  Node pittsburgh("Pittsburgh");
4  Node nyc("New York City");
5  Node nj("New Jersey");
6  demo: + portland + pittsburgh ->2566 portland ->2566 pittsburgh +
  nyc [800->800] pittsburgh, portland [2900<-2900] nyc, +
  nj <-13 nyc <-13 nj [2899<-2899] portland,
7  nj [790<-790] pittsburgh;
8  print(demo);
9
Nodes:
Node (0) : Portland --- Node (1) : Pittsburgh --- Node (2) : New York City
Node (3) : New Jersey

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

- **Graphs**
  - `graph.weight[node1, node2]`
  - `graph.num_nodes`
  - `graph.num_neighbors[A]`
  - `graph.node[n]`
  - `graph.neighbor[A, n]`

- **Nodes**
  - `v.name`
  - `v.curr_dist`
  - `v.visited`

- **Strings**
  - `string.length`
YAGL's Standard Libraries

- **stdgraph.ygl**
  - Graph copy_graph_lib(Graph g)
  - Graph reverse_graph_lib(Graph g)
  - void print_graph_lib(Graph g)

- **stdalgo.ygl**
  - void dfs(Graph G, Node vertex, int depth)
  - Node get_first_node_at_depth(Graph G, Node vertex, Node break, int depth)

*Used in upcoming demo!*
Node get_first_node_at_depth(Graph g, Node vertex, Node b, int depth) {
    reset(g);
    return get_first_node_at_depth_helper(g, vertex, b, depth);
}

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

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

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

    while (curr < size) {
        Node current;
        current = g.neighbor[vertex, curr];
    }

    if (current.curr_dist == 0) {
        current.curr_dist = vertex.curr_dist + 1;
    }
    curr = curr + 1;
}

Node new; /* Just a place holder */
while (curr < size) {
    Node current;
    current = g.neighbor[vertex, curr];
    new = get_first_node_at_depth_helper(g, current, break, depth);
    if (new == break) {
    } else {
        return new;
    }
    curr = curr + 1;
}
return new;
Built-in Functions

- **Graph Functionality**
  - `make_graph(int size)`
  - `insert_node(struct Graph *, struct Node *)`
  - `make_node(char *name)`
  - `get_neighbor(struct Graph *, struct Node *)`
  - `print_graph(struct Graph *)`
  - `insert_edge(struct Graph *, struct Node *, int, struct Node *)`
Who did what?

- First half through Hello World: Together
- Second half: Distributed Feature Development
  - From scanner → ... → codegen → tests
- Weekly meeting to merge code & features
- Process worked very well: all wanted to learn about the entire compiler!
Amazing Demo Time

Buckle up.
Thank you!