A GRAphe JaArgon

Zachary Salzbank
Erica Sponsler
Nathaniel Weiss
AGRAJAG’s Purpose

• C-based language

• Includes built-in Node data structure

• Ideal for storing graphs/trees

• Data processing
How to use AGRAJAG

• AGRAJAG’s syntax is extremely similar to C’s
  – Root method instead of main method
  – No easy pointer access
  – No For loops
  – No Strings implemented
    • Functionality is there: nodes of chars.
  – NODES
    • Created with Node<type> = <instance of type>;
What is a Node?

• A node can have a type of either a base type, or another node.
  – Can have Node<Node<Node<...>>> as many times as you would like, as long there is a base type.

• Acceptable base types:
  – Int
  – Char
  – Boolean

• Nodes can have up to 10 children
Examples

```java
void root()
{
    Node<Node<int>> x;
    Node<int> y;
    x = <<42>>;
    y = x.value;
    print(x.value.value == y.value);
}
```

```bash
$ ./agrajag < ./tests/test-node-nested-child-value2.ag 
true
```
Examples

```java
public class Main {

    public static void main(String[] args) {
        Node<Integer> root = root();
        System.out.println(search(root, 4));
    }

    public static Node<Integer> root() {
        Node<Integer> treeRoot = new Node<>(5);
        treeRoot.left = new Node<>(3);
        treeRoot.right = new Node<>(7);
        treeRoot.left.left = new Node<>(2);
        treeRoot.left.right = new Node<>(4);
        treeRoot.right.left = new Node<>(6);
        treeRoot.right.right = new Node<>(8);
        return treeRoot;
    }

    public static boolean search(Node<Integer> node, int searchFor) {
        while (node != null) {
            if (searchFor < node.value) {
                node = node.left;
            } else if (searchFor > node.value) {
                node = node.right;
            } else {
                return true;
            }
        }
        return false;
    }

}
```
Implementation

- Input
  - Scanner/Parser
    - AST
  - Semantic Checker
    - SAST
    - SAST
  - Compiler
    - Bytecode
  - Interpreter
    - Output
  - Bytecode Executer
    - Output
Nodes

- Variables are pointers to objects on "Node Heap"
- Node Heap stores values and pointers to children

```cpp
int x;
int y;
Node<int> z;

x = 3;
y = 5;
z = <4>;
z[0] = <2>;
```
Bytecode

• Node heap requires new instructions:
  – Ldh – Fetch from heap
  – Sth – Store to heap
  – Cnd – Create node on heap

• Different types differentiate values on the stack between pointers to nodes, and base types.
Lessons Learned

• Teamwork is essential

• Testing implementation details ahead of time

• Project perspective