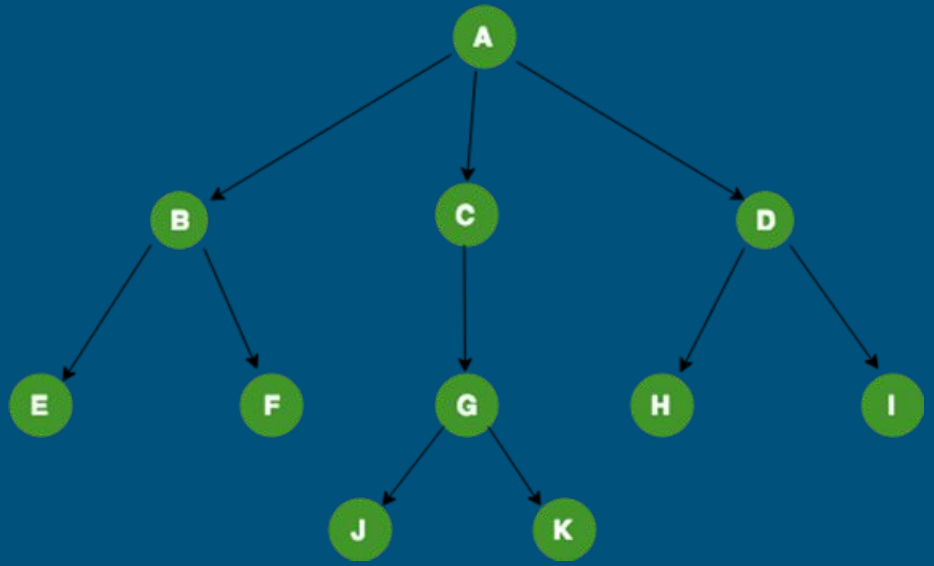
ARBOL

A Tree Language

Andrea Mary McCormick {amm2497}
Anthony Palmeira Nascimento {asp2199}
Derek Hui Zhang {dhz2104}

Table of Contents

1. ARBOL Overview
2. Compiler Architecture
3. Type System and Node Semantics
4. Built-In Tree Functions
5. Unit Testing CLI Tool
6. Demonstrations
7. Future Work



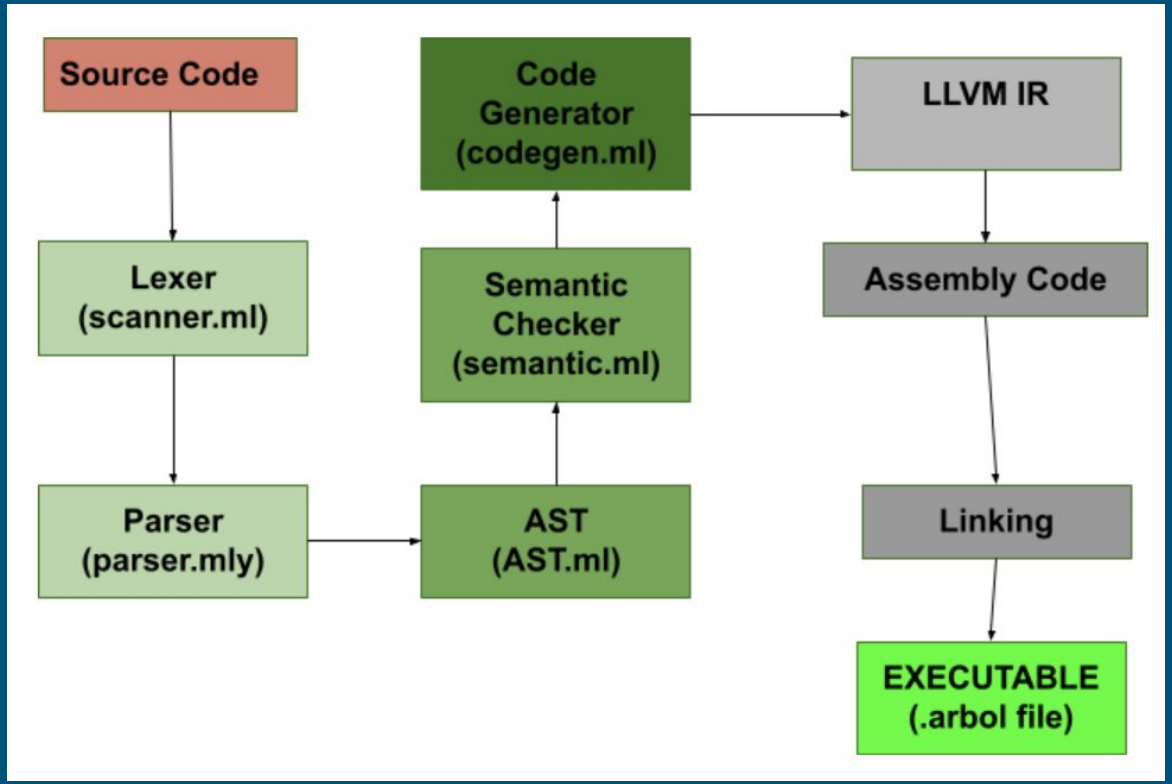
ARBOL

A Tree Language

Overview

- C-based language
- Imperative programming
- Sequential execution of instructions
- Strong Static-Typing

Compiler Architecture



Type System and Node Semantics

- Primitive Types: **`Integer, Float, Boolean, Char, String`**
- Derived recursive data type: **`Node`**

```
int @node => 5;

{
     V_type: int;
     V_name: "node";
     v_val:  5;
     node_val: true;
}
```

```
int var = 5;

{
     V_type: int;
     V_name: "var";
     v_val:  5;
     node_val: false;
}
```

Type System and Node Semantics

- Node operators:

Node Initialize	@
Node Dereference	~
Node Assignment	=>
Get Left Child	[
Get Right Child	]
Set Left Child	<--
Set Right Child	-->

Type System and Node Semantics

- Node operators demo:

```
function void main() {
    string @a => "a";
    string @b => "b";
    string @c => "c";
    a <-- c;
    a --> b;
    if (get_leftchild(a) == "c") {
        print("success");
    }
}

function string get_leftchild(string @t) {
    string @x = [t;
    return ~x;
}
```

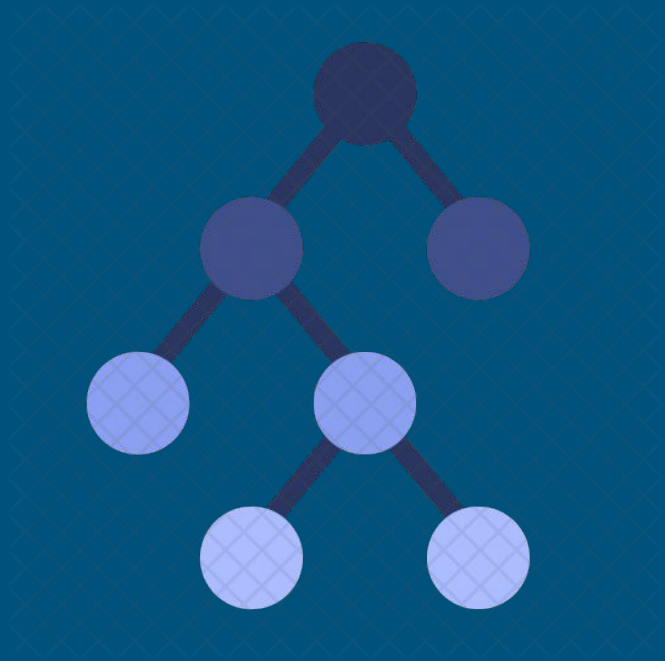
node1.arbol

node5.arbol

```
function void main() {
    int x = 5;
    int @b => 6;
    int @a => ~b;
    if ((~a) == 6) {
            print("success");
    }
}
```

Functions

- `void preorder(string @t)`
- `void inorder(int @t)`
- `void postorder(float @t)`
- `int height(int @t)`
- `void levelorder(char @t)`
- `void printlevel(string @t, int level)`



Unit Testing with Python CLI Tool

```python
def run_test_file(test_file):
    test_name = test_file[:-6]

    cmd = """
       ./arbol -c ./tests/{test_file} > {test_name}.ll
        /usr/local/opt/llvm@11/bin/llc -relocation-model=pic {test_name}.ll
> {test_name}.s
         /usr/bin/cc -c -DBUILD_TEST print.c
         /usr/bin/cc -o {test_name}.exe {test_name}.s print.o
         ./{test_name}.exe > {test_name}.out
    """.format(test_file=test_file, test_name=test_name)
```

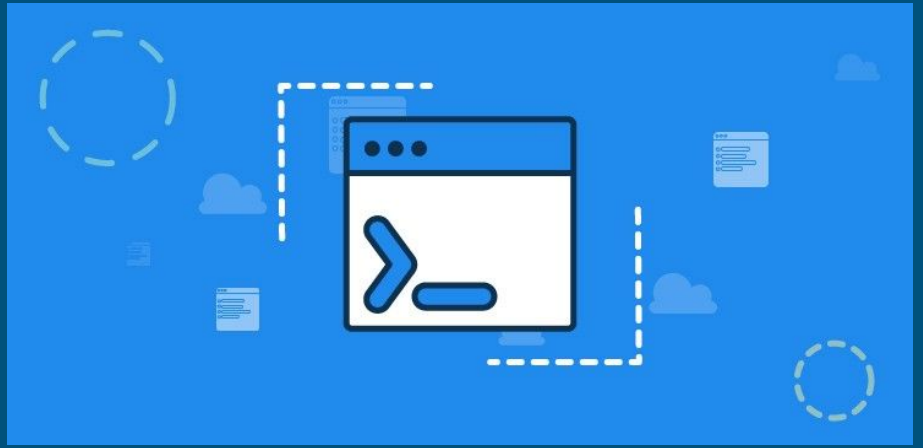
cli.py

```
`test-float1` Actual: ['3.141593'] | Expected: ['3.141593'] PASS
`test-arith2` Actual: ['24'] | Expected: ['24'] PASS
`test-gcd1` Actual: ['2', '3', '11'] | Expected: ['2', '3', '11'] PASS
`test-node2` Actual: ['5'] | Expected: ['5'] PASS
`test-func2` Actual: ['2'] | Expected: ['2'] PASS
`test-node6` Actual: ['3.420000'] | Expected: ['3.420000'] PASS
`test-func6` Actual: ['115'] | Expected: ['115'] PASS
`test-func4` Actual: [] | Expected: [] PASS
`test-node4` Actual: ['b'] | Expected: ['b'] PASS
```

fixtures.json

Demonstration

- *test-preorder2.arbol*
- *test-inorder1.arbol*
- *test-postorder1.arbol*



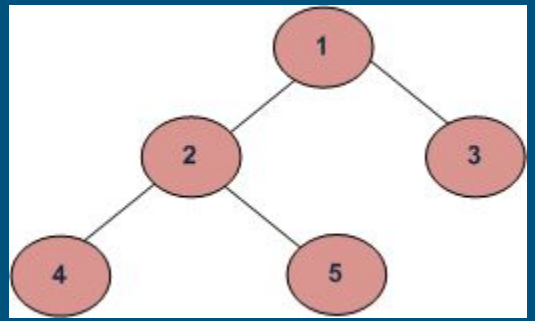
Traversals

test-postorder.arbol

```
function void main() {
    int @a => 1;
    int @b => 2;
    int @c => 3;
    int @d => 4;
    int @e => 5;

    a <-- b;
    a --> c;
    b <-- d;
    b --> e;

    postorder_int(a);
}
```



test-preorder2.arbol
test-inorder1.arbol
test-postorder1.arbol

```
========== ARBOL Tests ==========
`test-preorder2` Actual: ['1', '2', '4', '5', '3'] | Expected: ['1', '2', '4', '5', '3'] PASS
(env) (base) Anthonys-MacBook-Pro:arbol_tree_language anthonypalmeira$ arbol_test -t test-inorder1.arbol
========== ARBOL Tests ==========
`test-inorder1` Actual: ['4', '2', '5', '1', '3'] | Expected: ['4', '2', '5', '1', '3'] PASS
(env) (base) Anthonys-MacBook-Pro:arbol_tree_language anthonypalmeira$ arbol_test -t test-postorder1.arbol
========== ARBOL Tests ==========
`test-postorder1` Actual: ['4', '5', '2', '3', '1'] | Expected: ['4', '5', '2', '3', '1'] PASS
```

Future Work

- `Node` of `Node` nested types
- Balancing Trees, `search` function, and Complex Algorithms (i.e. Red-Black Algorithm)

Thank You!

Any Questions?