PLTree

A tree programming language

Overview

Philosophy: Everything is a tree

All data structures are built on the tree

A primitive type is a tree with a single node at the root and no leaves

A string is a tree of characters

A function is a tree of statements

Goal: Make it easy to create and work with trees.

Language compiles to the **C programming language**.

Basics

- Types: Integers, Doubles, Characters
- Booleans are represented by Integers
- Pseudo-types: String, Any

Declaration:

- int a 5; a = 6;
- char foo {‘a’} [42 17];

Control Flow:

```
if: 1 > 2 [ return:foo; ]
          else [ return:2; ]
```

Unique Operators:

- Accessor: foo->0;
- Width: int w #foo;

Functions:

```
bar : any arg [
          return:5;
   ]
```

Import: $filename$

File extension: .tree

Hello, World!

A simple “Hello, World!”

Code:

```
$stdio.tree$

string str “hello\n”;

print : str;
```

Output:

```
hello
```

Equivalent to:

Code:

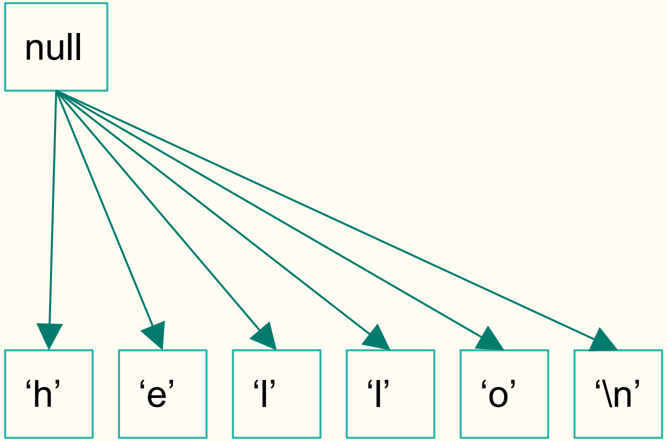
```
$stdio.tree$

string str ['h' 'e' 'l' 'l' 'o' '\n'];

print : str;
```

Output:

```
hello
```

Generated code

```
int main(int argc, char **argv) {
;
;
            struct tree * str = void_treemake(
            char_treemake('h', NULL),
            char_treemake('e', NULL),
            char_treemake('l', NULL),
            char_treemake('l', NULL),
            char_treemake('o', NULL),
            char_treemake('\n', NULL),
            NULL); inc_refcount(str);;
            print(
            str);
            dec_refcount(str);
            return 0;
}
```



The ‘print’ function

- Recursive
- Pre-Order Depth First Search
- Uses c function put_t

```
print: any data [
        int n #data;
        int i 0;

        put_t:data;

        i = 0;

        while: i < n [
        print:data->i;
        i = i + 1;
        ]

        return:data;
]
```

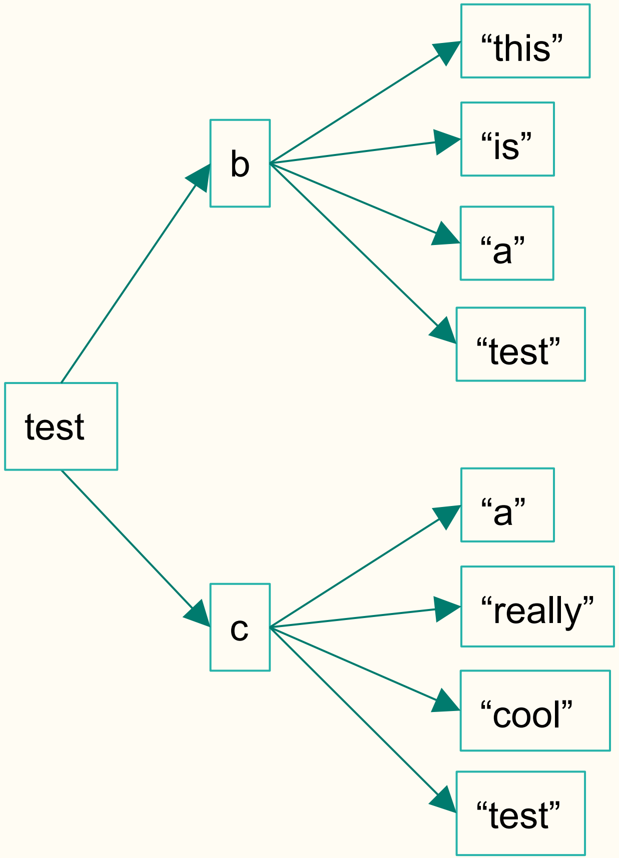
Example

Code:

```
string b ["this" "is" "a" "test"];
string c ["a" "really" "cool" "test"];
string test [b c];


print : [test->0->0 test->0->1 test->1->0 test->1->1 test->1->3 test->1->3];
```

Output:

```
thisisareallytesttest
```



Example

Code:

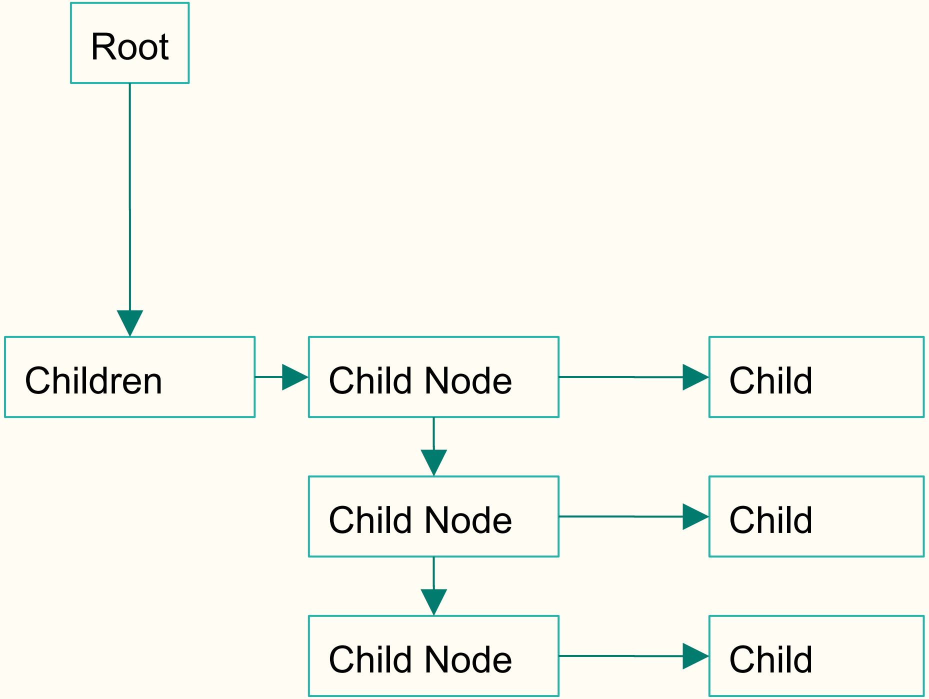
```
int test_tree {0} [1 2 3
                                     [4 5 6]
                             7
                                     [8
                                             [9 10]
                                     11]
                             12];

pretty_print:[0 test_tree];
```

Output:

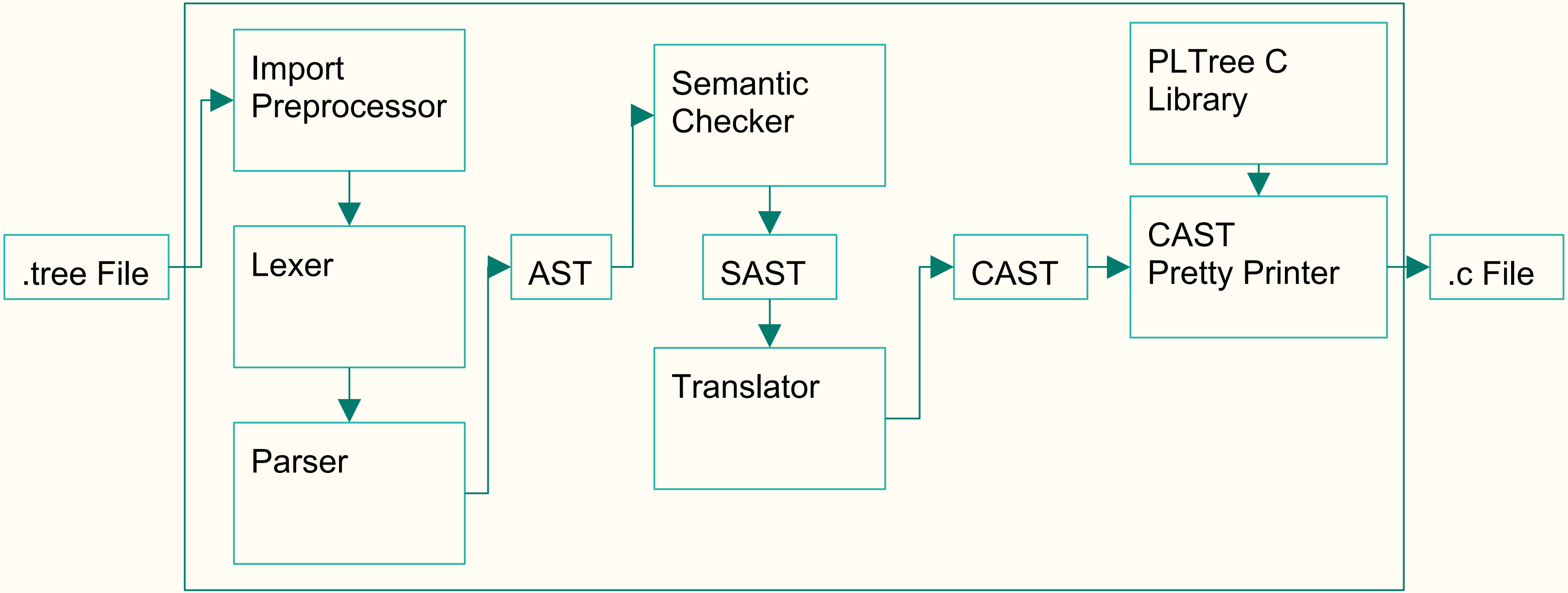
```
0
    1
    2
    3

        4
        5
        6
    7

        8

            9
            10
        11
    12
```

C Backend

```
struct tree {
                data_type type;
                union data_u data;
                int width;
                int refcount;
                struct List *children;
};
struct tree *treemake(
        data_type type,
        union data_u data,
        struct tree *child,
        va_list args);

struct tree*
inc_refcount(struct tree *t);

struct tree*
dec_refcount(struct tree *t);
```



Compiler structure

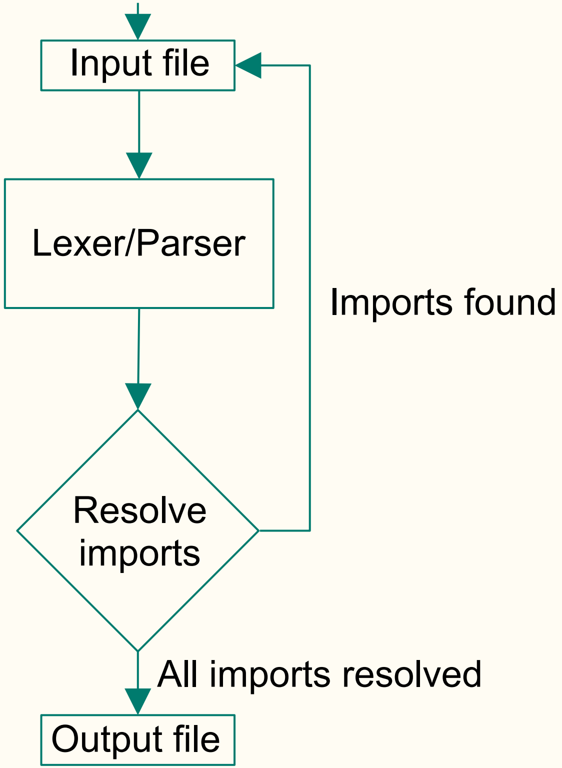


Import Preprocessor

Resolve all imports

$filename$ replaced with contents of filename

Prevent double imports by maintaining list of already imported files

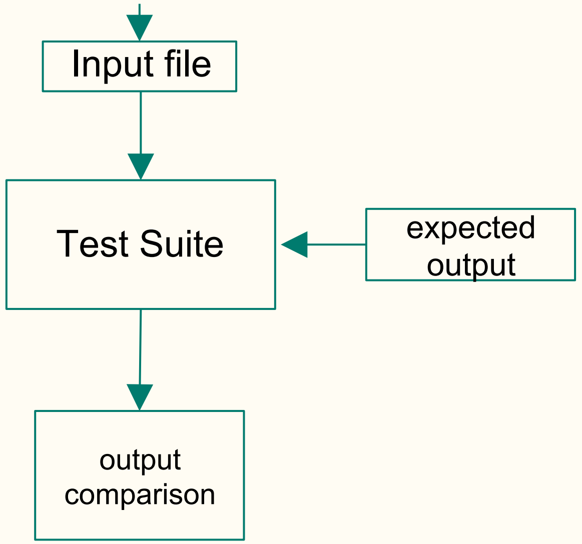


Test Suite

Managed by a bash script

Tests a .tree program’s output to ensure proper language behavior

Initially tested AST of a program



Testing

```
$ ./tester.sh -c tests/programs
tests/programs/fact: SUCCESS
tests/programs/fibo: SUCCESS
tests/programs/func_test: SUCCESS
tests/programs/gcd: SUCCESS
tests/programs/hello: SUCCESS
tests/programs/pretty_tree: SUCCESS
tests/programs/printing: SUCCESS
tests/programs/stdio: SUCCESS
```

Expected output of gcd.tree:

```
Testing iterative gcd with 65 and 195
65
Testing recursive gcd with 14 and 21
7
```

Generated output of gcd.tree:

```
Testing iterative gcd with 65 and 195
65
Testing recursive gcd with 14 and 21
7
```