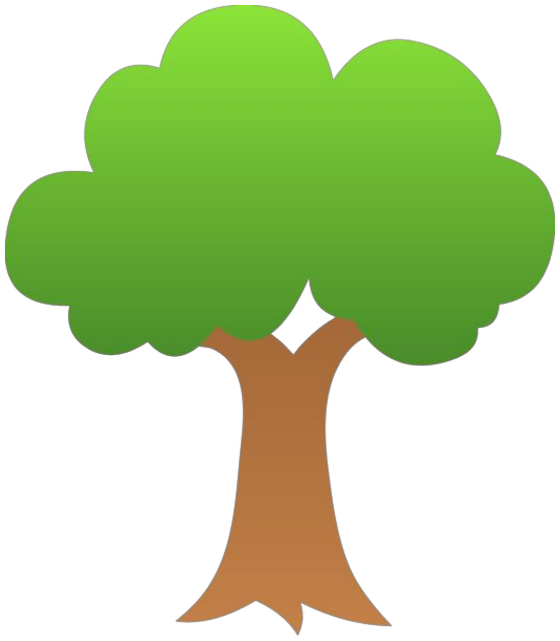
Jacob Graff - jag2302
Justin Walters - jw3043
Luis “Bert” Ramirez - lar2195
Shruti Kulkarni - sgk2118

Proposal: PLTree

A Tree Programming Language



LANGUAGE OVERVIEW/DESCRIPTION

We propose a language for usage and manipulation of trees where the main data type is a tree. Every variable will be treated as a tree; for example, a string in our language would be a tree with leaves of characters. The language will make it easy to create and edit trees with functions such as adding a new item at a certain position in the tree or deleting items. It will also make it simple to manipulate trees with common tree functions such as pruning, grafting, finding the root, and searching for an item. We provide functions such as `dfs` and `bfs` that will parse the tree in the appropriate order, applying a given function to each node. Additionally, the language will include templates for common tree structures, such as binary search trees, AVL trees, and heaps. These templates will make the language flexible enough to fit most users’ needs.

TYPICAL USE CASES

This programming language can be used to quickly create and manipulate trees. Common tree operations will be easy to use and understand. This language could be very useful for understanding of trees, parsing large data sets, and academic use. Practical use cases include simple creation and search of binary trees, or holding hierarchical data or sorted lists of data.

PARTS OF THE LANGUAGE

Built-In Types

Literals
boolean (true/false)
integer values (..., -3, -2, -1, 0, 1, 2, 3, …)
double values (3.5, 300.2, -4.5)
character values (‘a’, ‘8’, ‘\n’)

Primitives
bool (boolean value)
int (integer value)
double (decimal value)
char (character)

Collections
tree (the main type to be used)

Keywords
void
return

Operators
Arithmetic (+, -, *, /)
Numerical relational (==, !=, <, <=, >, >=)
Logical (&&, ||, !)

Control Flow
if (if/else statement)
while (while loop)

Declaring Variables
Variables are declared in the following syntax:
```
      (type name literal_value)
```
For example:
```
      (int a 5)
      (char b ‘h’)
      (double c 8.8)
      (tree t ()) /*empty tree*/
```
There are also unnamed variables, declared as such:
```
      (int 5)
      (char ‘B’)
```

Type Inference

Some types can be written as is, instead of with a type declaration.
For example:

```
      (int 5)  can be written as  5
      (char ‘a’)  can be written as  ‘a’
      (double 3.5)  can be written as  3.5
      (int a 5)  can be written as  (a 5)
```

In addition, strings, which are really trees of `char`, can be declared in the following manner:

```
      (tree ‘h’ ‘e’ ‘l’ ‘l’ ‘o’)  can be written as  “hello”
```

Built-In Functions
Examples include:

```
(bool and (bool a) (bool b))
(bool or (bool a) (bool b))
(bool equals (node a) (node b))

/* apply function to nodes of trees */
(void dfs (tree) (function)) /* depth-first search */
(void bfs (tree) (function)) /* breadth-first search*/

(void output (tree t)) /* writes a given leaf to the console */
(bool isleaf (tree t))
(tree branch (tree parent ((tree t) (int i))) /* returns the ith
                                                 branch of tree t
                                             */
```

User-Defined Functions
All functions accept as arguments exactly one value, a tree, which may itself contain additional arguments. A function is declared in the following syntax:

```
(type name (tree t) (
      /* do something */
      (return value)
))
```

Where `type` is the return type, `name` is the name of the function, `(tree t)` is the input argument, and the last parens `( /*do something */ )` are an execution statement.

In addition to the above, a **Standard Library** shall be included.

SAMPLE PROGRAM A

```
/* print implementation */
(void print (tree instring) (

     /* a function to be passed to dfs */
     (void accept (tree node) (
          (if (isleaf node) (
                    (output node)
               )
          )
     ))
     /* the instring tree is passed to dfs, along with the function
     accept. Every node in instring is passed to accept, which
     processes and outputs to the console if needed */
     (dfs (instring accept))
))

/* an example of type inference */
(print “Hello, World!\n”)
```

SAMPLE PROGRAM B

```
/* factorial function */
(int factorial (tree (int n)) (
     (if (n > 0)
          (return (n * (factorial n - 1)))
          (return 1)
     )
))
```