CS 472 Fall 2011
Project 2.2

Colby Blair

Due October 17th, 2011

Abstract

In computer science, one area of study is that of optimizing functions. There are many methods for optimization, and this repor will talk about Genetic Programming (GP). Genetic Programming creates mathematical expression trees, and modifies them to make educated guesses. They are useful for finding the function defintions for curves on a graph.

This report presents a GP with mathematical non-terminal symbols '+', '-', '*', and '/', and terminal values as contants and variables. Although very simple, this report setups a proof of concept GP for later reports. It talks about the crossover and selection functions, as well as the population representation. It also shows two trees before and after crossover, a sample tree over different values, and finally, the code used.

Contents

List of Figures

Part I
Representation Description

GPs are used to try to approximate a mathamatical expression **tree** (Section 1) that describes a function on a graph. In order to improve the approximation, a random set of expression trees are generated. This set is called the **population** (Section 2).

1 Trees

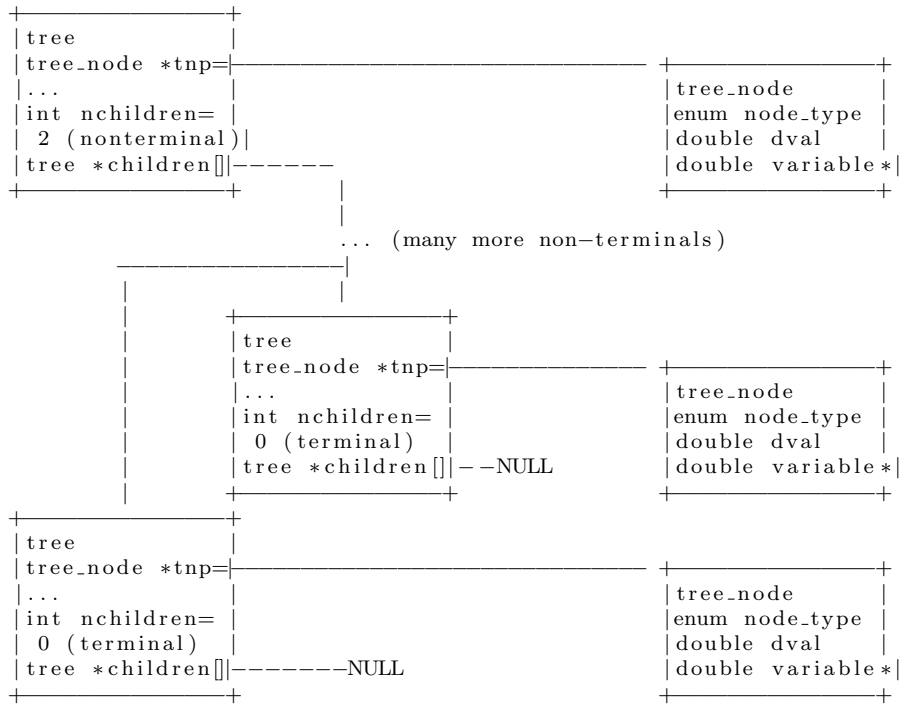


Figure 1: An expression tree (one per individual)

A tree is simply class, that has pointers to child trees. Since our operators ('+', '-', '*', and '/') only take a left hand and right hand expressions, each

tree only needs at most 2 children. But more or less can be inserted for future operators, on a per-operator basis. Since a tree simply points to other subtrees, the term **tree** in this report can mean either the whole tree or a subtree.

Our operators are called **non-terminals**, since they rely on the results of child subtrees to compute their results. Our **terminals** then are either constants or pointers to elements in a variable array (double, or decimal, values). Both are initialized randomly from their respective sets.

Each tree class instance points to a tree_node class. This class holds the enumerable type of the tree class; either 'plus', 'minus', 'multi', or 'div' for non-terminal trees (operators), or 'tree_double' or 'tree_variable' for terminal trees.

The terminal trees will be, in future projects, mutated using point mutation, but for now are left alone. The non-terminal trees are mutated by simple regenerating a random tree in place, and selected at random. Trees of type tree_var are, again, pointers to a variable array. This tree's value is initialized to point to a random element in the variable list. Since they are pointers, modifying variable values takes immediate affect throughout the tree. The variables in the variable array can be modified, and the tree evaluation and fitness functions (re)ran.

2 Population

In order to optimize lots of trees to reach an approximate solution

Part II
Functions and Generators

3 Fitness Function

$$
\begin{aligned}
f_i(expected) &= \text{Error} \\
&= |eval_i() - exptected| \\
&\text{where} \\
&eval_i() \text{ is the evaluation function in Figure 3}
\end{aligned}
$$

Figure 2: Fitness function

$$
\begin{aligned}
eval_i() &= \sum_{x=1}^{n} eval_{child_x}() \text{ if } i \text{ type is 'plus'} \\
&= eval_{child_1}() - eval_{child_2}() - ...eval_{child_n}() \text{ if } i \text{ type is 'minus'} \\
&= \prod_{x=1}^{n} eval_{child_x} \text{ if } i \text{ type is 'plus'} \\
&= eval_{child_1}()/eval_{child_2}()/...eval_{child_n}() \text{ if } i \text{ type is 'div'} \\
&= i_{constantvalue} \text{ if } i \text{ type is 'tree\_double'} \\
&= i_{variablevalu} \text{ if } i \text{ type is 'tree\_var'} \\
&\text{where} \\
&n \text{ is the number of children (0 or 2 only for now)}
\end{aligned}
$$

Figure 3: Evaluation function

4 Random Tree Generator

```
if random_value in 0...9 equals 0 or at depth 0:
        set this subtree to a terminal type; a randomly a contant or
                a variable
else:
        set this subtree to a nonterminal type; randomly a 'plus',
                'minus','multi','div'
        create each child from rand_tree_generator(depth - 1)
```

Figure 4: Random tree generator

The random tree generator generator is a recursive function that tries to construct a semi-inbalanced tree. There is a 1 in 10 chance that it will make any node not at the max depth a terminal, which leads to a moderately imbalanced tree. See Section 8 for a resulting tree.

```
n = the nth non-terminal to mutate
sum = how many terminals we've seen so far (global)
depth = depth of new tree
if current branch is a non-terimanal:
        sum++
if n is equal to sum:
        set this branch to a new tree using rand_tree_generator(depth)
        return
else:
        for each child in current branch:
                mutate_nth_nonterm(n, child)
return
```

Figure 5: The non-terminal mutation function (see Figure 4 for rand_tree_generator())

5 Mutation Function

The mutation function first calculates how many non-terminals are in the tree, and then starts counting the ones it has already seen. When it sees the nth non-terminal, it mutates it in place, and returns out.

6 Samples

6.1 A test individual

```
0:minus = -0.759484
 1:tree_var = 0.3
 1:multi = 0.459484
  2:multi = 0.0220245
   3:div = 1.77934e-05
    4:multi = 134.636
     5:div = 12.4104
      6:multi = 0.00322311
       7:tree_double = 0.056702
       7:tree_double = 0.0568429
      6:div = 25
       7:tree_var = 0.2
       7:tree_var = 0.2
     5:div = 10.8487
      6:plus = 0.257539
       7:tree_double = 0.0575395
       7:tree_var = 0.2
      6:plus = 0.357915
       7:tree_double = 0.0579151
       7:tree_var = 0.3
    4:plus = 417.426
     5:div = 416.667
      6:multi = 0.04
       7:tree_var = 0.2
       7:tree_var = 0.2
      6:multi = 0.06
       7:tree_var = 0.3
       7:tree_var = 0.2
     5:minus = 0.759496
      6:minus = -0.259496
       7:tree_var = 0.2
       7:tree_double = 0.0594961
      6:minus = -0.5
       7:tree_var = 0.3
```

```
      7:tree_var = 0.2
  3:minus = 1237.79
   4:minus = 9.5315
    5:plus = 0.377885
     6:plus = 0.5
      7:tree_var = 0.3
      7:tree_var = 0.2
     6:minus = -0.122115
      7:tree_double = 0.0608813
      7:tree_double = 0.0612335
    5:multi = -9.90939
     6:minus = -0.12325
      7:tree_double = 0.0615544
      7:tree_double = 0.0616953
     6:div = 80.4009
      7:tree_var = 0.2
      7:tree_double = 0.0621883
   4:minus = -1247.32
    5:plus = -0.4
     6:minus = -0.6
      7:tree_var = 0.3
      7:tree_var = 0.3
     6:tree_var = 0.2
    5:div = 1247.72
     6:tree_double = 0.0631197
     6:multi = 0.0126975
      7:tree_var = 0.2
      7:tree_double = 0.0634875
2:multi = 20.8624
 3:div = -388.287
  4:div = 0.199488
   5:tree_var = 0.2
   5:plus = 25.0641
    6:tree_double = 0.0641371
    6:div = 25
     7:tree_var = 0.2
     7:tree_var = 0.2
  4:div = -0.0129101
```

```
      5:plus = 233.178
       6:plus = 0.5
        7:tree_var = 0.3
        7:tree_var = 0.2
       6:div = 232.678
        7:tree_double = 0.0654911
        7:tree_double = 0.0656241
      5:minus = -0.332187
       6:tree_double = 0.0658589
       6:plus = 0.266328
        7:tree_var = 0.2
        7:tree_double = 0.0663285
   3:div = -0.0537295
    4:tree_var = 0.3
    4:minus = -62.0392
     5:minus = -0.0602061
      6:multi = 0.04
       7:tree_var = 0.2
       7:tree_var = 0.2
      6:multi = 0.0202061
       7:tree_double = 0.0673537
       7:tree_var = 0.3
     5:div = 62.0994
      6:multi = 0.06
       7:tree_var = 0.3
       7:tree_var = 0.2
      6:plus = 0.268387
       7:tree_var = 0.2
       7:tree_double = 0.0683868
Tree has 51 terminal(s).
Tree has 50 non-terminal(s).
Tree fitness: 7.66248
```

6.2 Another test individual

```
0:div = 0.410738
 1:tree_double = 0.0496583
 1:div = 49.0279
```

```
  2:div = 0.387529
   3:multi = 12.9023
    4:minus = -0.5
     5:tree_var = 0.3
     5:tree_var = 0.2
    4:multi = -25.8045
     5:minus = -97.8877
      6:plus = 0.350965
       7:tree_var = 0.3
       7:tree_double = 0.0509653
      6:div = 97.5368
       7:tree_double = 0.0512627
       7:tree_var = 0.2
     5:div = 0.263614
      6:multi = 0.0103527
       7:tree_double = 0.0517636
       7:tree_var = 0.2
      6:div = 366.419
       7:tree_double = 0.0521628
       7:tree_double = 0.0523193
   3:tree_var = 0.2
  2:tree_double = 0.0526323
Tree has 13 terminal(s).
Tree has 12 non-terminal(s).
Term mutation on 10 terminal
Tree fitness: 6.49226
```

7 Mutation

The mutation is still a bit problematic due to some memory issues, but when functions properly, looks like this:

```
0:multi = -0.0400978
 1:plus = -34.4194
  2:multi = -34.4489
   3:minus = -0.210426
    4:multi = 0.284427
     5:minus = -195.359
      6:div = 195.087
       7:tree_double = 0.0715095
       7:tree_double = 0.0716817
      6:plus = 0.272143
       7:tree_var = 0.2
       7:tree_double = 0.0721435
     5:multi = -0.00145592
      6:minus = -0.272707
       7:tree_var = 0.2
       7:tree_double = 0.072707
      6:multi = 0.00533878
       7:tree_double = 0.0729809
       7:tree_double = 0.0731531
    4:div = -0.0740013
     5:minus = -181.236
      6:plus = 0.147637
       7:tree_double = 0.0736853
       7:tree_double = 0.0739514
      6:div = 181.088
       7:tree_double = 0.0742253
       7:tree_double = 0.0743975
     5:tree_double = 0.0745618
   3:plus = 163.71
    4:div = 0.0192317
     5:minus = 0.3
      6:minus = -0.5
       7:tree_var = 0.2
```

```
      7:tree_var = 0.3
     6:tree_var = 0.2
    5:plus = 173.325
     6:div = 173.265
      7:tree_double = 0.0758845
      7:tree_double = 0.0760567
     6:multi = 0.06
      7:tree_var = 0.3
      7:tree_var = 0.2
   4:plus = 163.691
    5:div = −0.0100379
     6:minus = −0.6
      7:tree_var = 0.3
      7:tree_var = 0.3
     6:div = 166.038
      7:tree_double = 0.0775202
      7:tree_double = 0.0776924
    5:plus = 163.701
     6:div = 163.685
      7:tree_double = 0.0780759
      7:tree_double = 0.078248
     6:multi = 0.0157388
      7:tree_var = 0.2
      7:tree_double = 0.0786941
2:plus = 0.0295132
 3:div = −0.0533446
  4:multi = −0.612537
   5:minus = −314.913
    6:div = 158.345
     7:tree_double = 0.0793829
     7:tree_double = 0.079555
    6:div = 156.567
     7:tree_double = 0.0798368
     7:tree_double = 0.0800011
   5:multi = 0.0019451
    6:tree_double = 0.0802751
    6:multi = 0.0242304
     7:tree_var = 0.3
```

```
        7:tree_double = 0.0807681
    4:multi = 30.604
     5:minus = −167.523
      6:div = 150.856
       7:tree_double = 0.0813316
       7:tree_double = 0.0815038
      6:div = 16.6667
       7:tree_var = 0.2
       7:tree_var = 0.3
     5:plus = −0.182686
      6:tree_var = 0.2
      6:minus = −0.382686
       7:tree_var = 0.3
       7:tree_double = 0.0826856
  3:tree_double = 0.0828578
1:multi = 0.00116498
 2:plus = −0.0299122
  3:div = −9.23216e−06
   4:minus = 141.059
    5:minus = −141.179
     6:plus = 0.167453
      7:tree_double = 0.0836404
      7:tree_double = 0.0838126
     6:div = 141.012
      7:tree_double = 0.0840943
      7:tree_double = 0.0843291
    5:div = 0.119847
     6:div = 139.066
      7:tree_double = 0.0847126
      7:tree_double = 0.0848848
     6:multi = 0.06
      7:tree_var = 0.2
      7:tree_var = 0.3
   4:plus = −767.882
    5:div = −784.636
     6:multi = 0.00737919
      7:tree_double = 0.0858161
      7:tree_double = 0.0859883
```

```
      6:minus = −0.172712
       7:tree_double = 0.0862701
       7:tree_double = 0.0864422
     5:plus = 16.7538
      6:div = 16.6667
       7:tree_var = 0.3
       7:tree_var = 0.2
      6:tree_double = 0.0871623
  3:multi = −0.0299029
   4:multi = −0.33349
    5:minus = −16.6745
     6:div = 16.6667
      7:tree_var = 0.3
      7:tree_var = 0.2
     6:multi = 0.00781573
      7:tree_double = 0.0883206
      7:tree_double = 0.0884928
    5:multi = 0.02
     6:plus = 0.5
      7:tree_var = 0.3
      7:tree_var = 0.2
     6:multi = 0.04
      7:tree_var = 0.2
      7:tree_var = 0.2
   4:tree_double = 0.0896667
2:div = −0.0389466
  3:multi = −0.0537658
   4:plus = −33.2422
    5:div = −33.3333
     6:multi = 0.06
      7:tree_var = 0.2
      7:tree_var = 0.3
     6:minus = −0.5
      7:tree_var = 0.2
      7:tree_var = 0.3
    5:tree_double = 0.0911537
   4:div = 0.0016174
    5:minus = −36.4563
```

```
    6:plus = 0.291717
     7:tree_double = 0.0917172
     7:tree_var = 0.2
    6:div = 36.1646
     7:tree_double = 0.0921712
     7:tree_var = 0.3
   5:minus = −16.9594
    6:plus = 0.292719
     7:tree_double = 0.092719
     7:tree_var = 0.2
    6:div = 16.6667
     7:tree_var = 0.2
     7:tree_var = 0.3
3:plus = 477.555
 4:div = 477.317
  5:multi = 0.0177303
   6:plus = 0.188052
    7:tree_double = 0.0939399
    7:tree_double = 0.0941121
   6:tree_double = 0.0942843
  5:multi = 0.118161
   6:plus = 0.4
    7:tree_var = 0.2
    7:tree_var = 0.2
   6:plus = 0.295403
    7:tree_var = 0.2
    7:tree_double = 0.0954034
 4:minus = 0.237644
  5:minus = −0.231942
   6:plus = 0.191942
    7:tree_double = 0.0958887
    7:tree_double = 0.096053
   6:multi = 0.04
    7:tree_var = 0.2
    7:tree_var = 0.2
  5:minus = −0.00570256
   6:tree_var = 0.2
   6:minus = −0.194297
```

```
        7:tree_double = 0.0970626
        7:tree_double = 0.0972348
Tree has 94 terminal(s).
Tree has 93 non-terminal(s).

Term mutation on 10 terminal
0:1
 1:2
  2:3
   3:4
    4:5
     5:6
      6:7
       7:7
       7:7
      6:8
       7:8
       7:8
     5:9
      6:10!mutating!
      6:11
    4:12
   3:13
  2:14
 1:15

After mutation:
0:multi = 3610.36
 1:plus = 3.09908e+06
  2:multi = 3.09908e+06
   3:minus = 18930.3
    4:multi = -18930.2
     5:minus = -195.359
      6:div = 195.087
       7:tree_double = 0.0715095
       7:tree_double = 0.0716817
      6:plus = 0.272143
       7:div = 19654.2
```

```
        8:div = 0.000358716
         9:multi = -12.2866
          10:plus = 0.293952
           11:div = 0.286747
            12:div = 11.1111
             13:tree_var = 0.3
             13:tree_var = 0.3
            12:plus = 0.313866
             13:tree_var = 0.2
             13:tree_double = 0.113866
           11:multi = 0.00720503
            12:multi = 0.0228827
             13:tree_var = 0.2
             13:tree_double = 0.114414
            12:plus = 0.314868
             13:tree_var = 0.2
             13:tree_double = 0.114868
          10:plus = -41.7979
           11:multi = -0.131222
            12:minus = -0.415525
             13:tree_var = 0.3
             13:tree_double = 0.115525
            12:plus = 0.315799
             13:tree_double = 0.115799
             13:tree_var = 0.2
           11:div = -41.6667
            12:multi = 0.04
             13:tree_var = 0.2
             13:tree_var = 0.2
            12:minus = -0.6
             13:tree_var = 0.3
             13:tree_var = 0.3
6:multi = 0.00533878
     7:tree_double = 0.0729809
     7:tree_double = 0.0731531
  4:div = -0.0740013
   5:minus = -181.236
    6:plus = 0.147637
```

```
      7:tree_double = 0.0736853
      7:tree_double = 0.0739514
     6:div = 181.088
      7:tree_double = 0.0742253
      7:tree_double = 0.0743975
    5:tree_double = 0.0745618
 3:plus = 163.71
  4:div = 0.0192317
   5:minus = 0.3
    6:minus = −0.5
     7:tree_var = 0.2
     7:tree_var = 0.3
    6:tree_var = 0.2
   5:plus = 173.325
    6:div = 173.265
     7:tree_double = 0.0758845
     7:tree_double = 0.0760567
    6:multi = 0.06
     7:tree_var = 0.3
     7:tree_var = 0.2
  4:plus = 163.691
   5:div = −0.0100379
    6:minus = −0.6
     7:tree_var = 0.3
     7:tree_var = 0.3
    6:div = 166.038
     7:tree_double = 0.0775202
     7:tree_double = 0.0776924
   5:plus = 163.701
    6:div = 163.685
     7:tree_double = 0.0780759
     7:tree_double = 0.078248
    6:multi = 0.0157388
     7:tree_var = 0.2
     7:tree_double = 0.0786941
2:plus = 0.0295132
 3:div = −0.0533446
  4:multi = −0.612537
```

```
      5:minus = −314.913
       6:div = 158.345
        7:tree_double = 0.0793829
        7:tree_double = 0.079555
       6:div = 156.567
        7:tree_double = 0.0798368
        7:tree_double = 0.0800011
      5:multi = 0.0019451
       6:tree_double = 0.0802751
       6:multi = 0.0242304
        7:tree_var = 0.3
        7:tree_double = 0.0807681
    4:multi = 30.604
     5:minus = −167.523
      6:div = 150.856
       7:tree_double = 0.0813316
       7:tree_double = 0.0815038
      6:div = 16.6667
       7:tree_var = 0.2
       7:tree_var = 0.3
     5:plus = −0.182686
      6:tree_var = 0.2
      6:minus = −0.382686
       7:tree_var = 0.3
       7:tree_double = 0.0826856
  3:tree_double = 0.0828578
1:multi = 0.00116498
 2:plus = −0.0299122
  3:div = −9.23216e−06
   4:minus = 141.059
    5:minus = −141.179
     6:plus = 0.167453
      7:tree_double = 0.0836404
      7:tree_double = 0.0838126
     6:div = 141.012
      7:tree_double = 0.0840943
      7:tree_double = 0.0843291
    5:div = 0.119847
```

```
      6:div = 139.066
       7:tree_double = 0.0847126
       7:tree_double = 0.0848848
      6:multi = 0.06
       7:tree_var = 0.2
       7:tree_var = 0.3
    4:plus = -767.882
     5:div = -784.636
      6:multi = 0.00737919
       7:tree_double = 0.0858161
       7:tree_double = 0.0859883
      6:minus = -0.172712
       7:tree_double = 0.0862701
       7:tree_double = 0.0864422
     5:plus = 16.7538
      6:div = 16.6667
       7:tree_var = 0.3
       7:tree_var = 0.2
      6:tree_double = 0.0871623
   3:multi = -0.0299029
    4:multi = -0.33349
     5:minus = -16.6745
      6:div = 16.6667
       7:tree_var = 0.3
       7:tree_var = 0.2
      6:multi = 0.00781573
       7:tree_double = 0.0883206
       7:tree_double = 0.0884928
     5:multi = 0.02
      6:plus = 0.5
       7:tree_var = 0.3
       7:tree_var = 0.2
      6:multi = 0.04
       7:tree_var = 0.2
       7:tree_var = 0.2
    4:tree_double = 0.0896667
  2:div = -0.0389466
   3:multi = -0.0537658
```

```
 4:plus = -33.2422
  5:div = -33.3333
   6:multi = 0.06
    7:tree_var = 0.2
    7:tree_var = 0.3
   6:minus = -0.5
    7:tree_var = 0.2
    7:tree_var = 0.3
  5:tree_double = 0.0911537
 4:div = 0.0016174
  5:minus = -36.4563
   6:plus = 0.291717
    7:tree_double = 0.0917172
    7:tree_var = 0.2
   6:div = 36.1646
    7:tree_double = 0.0921712
    7:tree_var = 0.3
  5:minus = -16.9594
   6:plus = 0.292719
    7:tree_double = 0.092719
    7:tree_var = 0.2
   6:div = 16.6667
    7:tree_var = 0.2
    7:tree_var = 0.3
3:plus = 477.555
 4:div = 477.317
  5:multi = 0.0177303
   6:plus = 0.188052
    7:tree_double = 0.0939399
    7:tree_double = 0.0941121
   6:tree_double = 0.0942843
  5:multi = 0.118161
   6:plus = 0.4
    7:tree_var = 0.2
    7:tree_var = 0.2
   6:plus = 0.295403
    7:tree_var = 0.2
    7:tree_double = 0.0954034
```

```
4:minus = 0.237644
 5:minus = -0.231942
  6:plus = 0.191942
   7:tree_double = 0.0958887
   7:tree_double = 0.096053
  6:multi = 0.04
   7:tree_var = 0.2
   7:tree_var = 0.2
 5:minus = -0.00570256
  6:tree_var = 0.2
  6:minus = -0.194297
   7:tree_double = 0.0970626
   7:tree_double = 0.0972348
```

8 Output

The eval for our trees isn't great, but we are not yet mutating or in any way trying to improve the fitnesses yet. For the following figures, values for x and y respectively were $(.2, .3)$, $(5, 7)$, and $(13, 20)$:

Figure 6: The expression tested against, $x^3 + 5y^3 - 4xy + 7$

Figure 7: Results from a random tree's expression

9 Code

9.1 Makefile

```
PROC=eval
CPP=g++
#CPPFLAGS=-g -pg -Wno-write-strings
#CPPFLAGS=-g -pg -Wno-write-strings -DDEBUG=1
CPPFLAGS=-g -pg -Wno-write-strings -DDEBUG_TREE=1
OBJS=tree_gp.o darray.o tree_node.o tree.o main.o test.o

all: $(OBJS)
        $(CPP) $(CPPFLAGS) $(OBJS) -o $(PROC)

main.o: main.cpp
        $(CPP) $(CPPFLAGS) main.cpp -c

tree_node.o: tree_node.cpp tree_node.h
        $(CPP) $(CPPFLAGS) tree_node.cpp -c

tree.o: tree.cpp tree.h
        $(CPP) $(CPPFLAGS) tree.cpp -c

test.o: test.cpp test.h
        $(CPP) $(CPPFLAGS) test.cpp -c

darray.o: darray.cpp darray.h
```

```
	$(CPP) $(CPPFLAGS) darray.cpp -c

tree_gp.o: tree_gp.cpp tree_gp.h
	$(CPP) $(CPPFLAGS) tree_gp.cpp -c

clean:
	rm $(PROC) *.o gmon.out
```

9.2 main.h

```
#ifndef _MAIN_H
#define _MAIN_H

#ifdef DEBUG
#define DEBUGMSG(arg) (cout << arg << endl)
#else
#define DEBUGMSG(arg) ;
#endif

#endif
```

9.3 main.cpp

```
#include <iostream>
#include "darray.h"
#include "tree_gp.h"

//#ifdef DEBUG
#include "test.h"
//#endif

using namespace std;

int main()
{
	//run no matter what for now
	//#ifdef DEBUG
	//test_nodes();
	//test_darray();
	//test_trees();
	//test_tree_copy();
	//test_tree_replace();
	//test_tree_crossover();
	//#endif
```

```
        //Main eval
        darray *dp1 = new darray(2, false);
        dp1->a[0] = .2;
        dp1->a[1] = .3;
        tree_gp *tgp1 = new tree_gp(10, 5, &dp1);
        //x^3 + 5y^3 - 4xy + 7
        //= (.2)^3 + 5(.3)^3 - 4(.2)(.3) + 7
        //= .008 + .135 - .24 + 7
        //= 6.903
        double dexpected = 6.903;

        for(int i = 0; i < 30; i++)
        {
                tgp1->ss(dexpected);
                tgp1->print_fitnesses(dexpected);
                int mini = tgp1->get_lowest_fitness_index(dexpected);
                cout << "Min fitness is " << mini << " element. "
                 << "Its eval value is " << tgp1->get_eval(mini) << endl;
                int mini2 = tgp1->get_second_lowest_fitness_index(dexpected);
                cout << "Second min fitness is " << mini2 << " element. "
                 << "Its eval value is " << tgp1->get_eval(mini2) << endl;
        }

        return(0);
}
```

9.4 tree_node.h

```
#ifndef _TREE_NODE_H
#define _TREE_NODE_H

#include <iostream>

#include "darray.h"

using namespace std; //for string

//how many types? see tree_node::node_type
// used in tree::gen_rand_node()
#define NTYPES 4

//how many terminal types? see tree_node::node_type
// used in tree_gen_rand_term_tree_node()
#define NTERMTYPES 2

class tree_node
```

```
{
//public enum here so private members can see
public:
        enum node_type
        {
                plus,
                minus,
                multi,
                div,
                tree_double,     //terminal
                tree_var,        //terminal
                null
        };

private:
        node_type ntype; //type of node (see node_type)
        double dval; //for tree_double types only
        int dpi;     //index the ddp points to in dp
        darray *dp; //darray pointer
        double *ddp; //double pointer to rand element in this->dp

public:

        tree_node(tree_node::node_type, double, darray**);
        bool copy(tree_node**);
        double get_dval();
        double get_ddp_val();
        tree_node::node_type get_ntype();

        bool set_ddp(int);

        bool print_ntype();
        bool print_dval();
        bool print_ddp();
        bool print_members();
};

#endif
```

9.5 tree_node.cpp

```
#include <iostream>
#include <stdarg.h>
#include <typeinfo>
#include <cstdlib>
```

```
#include "tree_node.h"
#include "tree.h"
#include "main.h"

using namespace std;

tree_node::tree_node(tree_node::node_type val, double dval, darray **dp)
{
        DEBUGMSG("DEBUG: tree_node.cpp: Setting node type");

        //init members to default vals
        this->dval = 0;
        this->dp = NULL;
        this->ddp = NULL;

        switch (val)
        {
                case tree_node::plus:
                {
                        this->ntype = val;
                        DEBUGMSG(" Node type == plus");
                        break;
                }
                case tree_node::minus:
                {
                        this->ntype = val;
                        DEBUGMSG(" Node type == minus");
                        break;
                }
                case tree_node::multi:
                {
                        this->ntype = val;
                        DEBUGMSG(" Node type == multi");
                        break;
                }
                case tree_node::div:
                {
                        this->ntype = val;
                        DEBUGMSG(" Node type == div");
                        break;
                }
                case tree_node::tree_double:
                {
                        this->ntype = val;
                        //get the float val
```

```
                this->dval = dval;
                DEBUGMSG(" Node type == tree_double");
                DEBUGMSG(" Node val == " << this->dval);
                break;
        }
        case tree_node::tree_var:
        {
                DEBUGMSG(" Node type == tree_var");

                this->ntype = val;
                //get the float val
                //get darray pointer from va_args
                //TODO: pass dp by reference instead

                /* initialize random seed: */
                srand ( clock() );

                //set dp to point to reference of passed in dp
                this->dp = (*dp);

                //set dpi
                /* generate secret number: */
                // select random element in dp
                this->dpi = rand() % this->dp->get_size();

                //set ddp to point to a random element of dp->a
                this->ddp = &this->dp->a[this->dpi];
                DEBUGMSG(" Node val from rand index " << j << "== " << *this
                break;
        }
        default:
                cerr << "ERROR: Node type not set, got val " \
                        << val << endl;
                exit(1);
    }
}


bool tree_node::copy(tree_node** to)
{
    switch (this->ntype)
    {
        case tree_node::tree_double:
        {
            (*to) = new tree_node(this->ntype, this->dval, NULL);
```

```
                        return(true);
                }
                case tree_node::tree_var:
                {
                        //TODO: not sure how stable this is exactly
                        (*to) = new tree_node(this->ntype, 0.0, &this->dp);
                        //TODO: set ddp to dp index
                        (*to)->set_ddp(this->dpi);
                        return(true);
                }
                default:
                {
                        (*to) = new tree_node(this->ntype, 0.0, NULL);
                        return(true);
                }
        }
}

double tree_node::get_dval()
{
        if(this == NULL)
        {
                return(NULL);
        }

        return(this->dval);
}


double tree_node::get_ddp_val()
{
        if(this == NULL)
        {
                return(NULL);
        }

        return(*this->ddp);
}


tree_node::node_type tree_node::get_ntype()
{
        if(this == NULL)
        {
                return(tree_node::null);
```

```
        }

        return(this->ntype);
}


bool tree_node::set_ddp(int i)
{
        if(i >= this->dp->get_size())
        {
                return(false);
        }

        this->dpi = i;
        this->ddp = &this->dp->a[i];

        return(true);
}


bool tree_node::print_ntype()
{
        if(this == NULL)
        {
                cout << "(!null!)";
        }
        else
        {
                switch (this->ntype)
                {
                        case tree_node::plus:
                        {
                                cout << "plus";
                                break;
                        }
                        case tree_node::minus:
                        {
                                cout << "minus";
                                break;
                        }
                        case tree_node::multi:
                        {
                                cout << "multi";
                                break;
                        }
```

```
                        case tree_node::div:
                        {
                                cout << "div";
                                break;
                        }
                        case tree_node::tree_double:
                        {
                                cout << "tree_double";
                                break;
                        }
                        case tree_node::tree_var:
                        {
                                cout << "tree_var";
                                break;
                        }
                } //end switch
        }
}


bool tree_node::print_dval()
{
        cout << this->dval;
        return(true);
}


bool tree_node::print_ddp()
{
        if(this->ddp == NULL)
        {
                cout << "   : ";
        }
        else
        {
                cout << *this->ddp << " : ";
        }

        return(true);
}

bool tree_node::print_members()
{
        this->print_ntype();
        cout << " : ";
```

```
        this->print_dval();
        cout << " : ";
        this->print_ddp();
        cout << "\n";
}
```

9.6 tree.h

```
#ifndef _TREE_H
#define _TREE_H

#include <time.h>

#include "tree_node.h"
#include "darray.h"


#ifdef DEBUG_TREE
#define DEBUG_TREE_MSG(arg) (cout << arg << endl)
#else
#define DEBUG_TREE_MSG(arg) ;
#endif


extern int SUM_TEMP;


#define MAX_CHILDREN 2

class tree
{
private:

public:
        //members
        tree_node *tnp;
        darray *dp; //darray pointer, for tree_double use only
        int nchildren;
        int depth;  //how deep the current tree is
        tree *children[MAX_CHILDREN];

        //methods
        tree(int, darray*);
        ~tree();
        bool copy(tree**);
        tree_node *gen_rand_nonterm_tree_node(darray*);//[non]terminal vals
```

```
        tree_node *gen_rand_term_tree_node(darray*); //terminal vals

        double eval();
        double fitness(double);

        bool is_term();
        bool is_nonterm();

        int count_terms();
        int count_nonterms();

        bool crossover(tree**,tree**);

        bool print(int);
        bool print_tnp_ntype();
};

//External tree stuff
bool mutate_nth_nonterm(tree**,int, int, int, darray*);
bool tree_replace_nth_nonterm(tree**,tree**, int);
bool tree_crossover(tree**,tree**);

#endif
```

9.7 tree.cpp

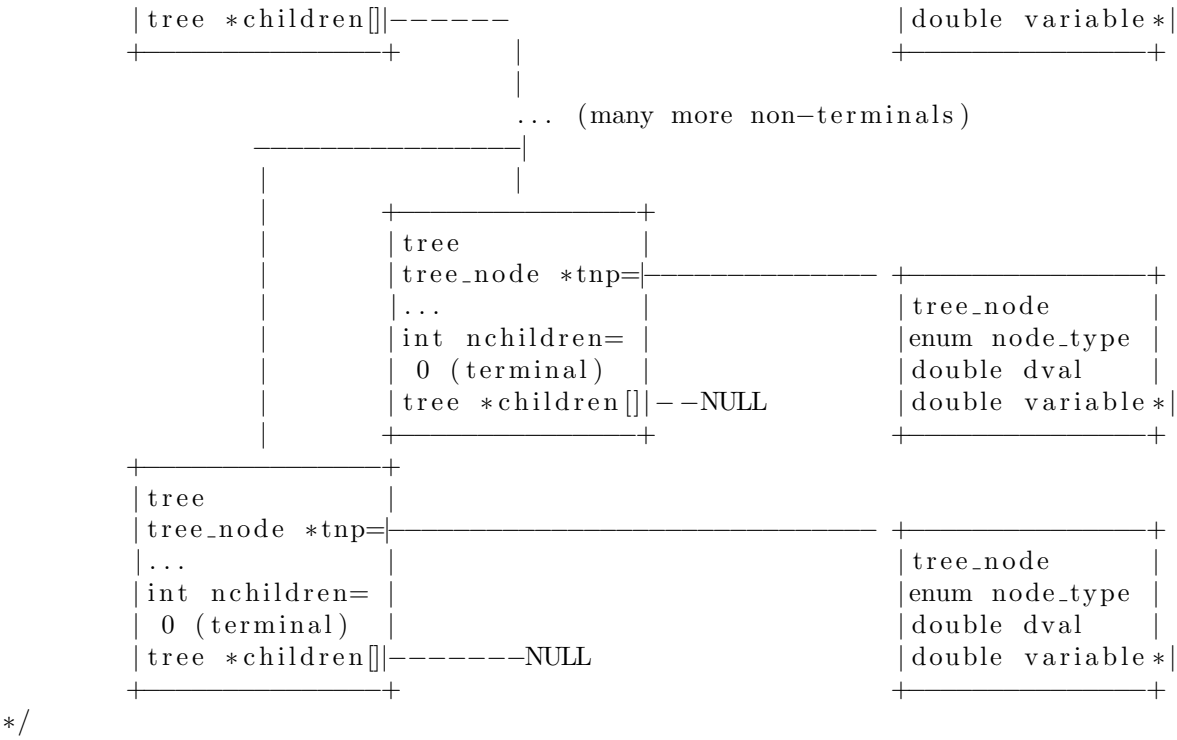
```
#include <time.h>
#include <iomanip>
#include <cmath>
#include <cstdlib>

#include "tree.h"
#include "tree_node.h"
#include "main.h"

int SUM_TEMP;

/*
This is the structure I am trying to represent

        +---------------+
        |tree           |
        |tree_node *tnp=|------------------------------- +---------------+
        |...            |                                |tree_node      |
        |int nchildren= |                                |enum node_type |
        | 2 (nonterminal)|                               |double dval    |
```




```
*/

////////////////////////////////////////////////////////////////////////////
//Tree
////////////////////////////////////////////////////////////////////////////

tree::tree(int depth, darray *dp)
{
        this->dp = dp;
        this->depth = depth;

        //init null children
        this->nchildren = 0;
        for(int i = 0; i < MAX_CHILDREN; i++)
        {
                this->children[i] = NULL;
        }

        //Terminal
        // if we've reached the bottom, or a random fraction of total nodes
        // be a terminal
        /* initialize random seed: */
        srand ( clock() );
```

```
        /* generate secret number: */
        int rand_val = rand() % 10; //0-9 values
        //cout << "DEBUG: tree.cpp: rand_val = " << rand_val << endl;
        // 1 out of 10 rand nodes get set to terminal
        bool rand_term = (rand_val == 0);
        if(depth <= 0 ) //|| rand_term == true)
        {
                this->gen_rand_term_tree_node(dp);
                return;
        }

        //Nonterminal
        this->gen_rand_nonterm_tree_node(dp);

        //create the children
        this->nchildren = MAX_CHILDREN;
        for(int i = 0; i < MAX_CHILDREN; i++)
        {
                DEBUGMSG("DEBUG: tree.cpp: Gen child " << i  << "at depth " << depth
                this->children[i] = new tree(depth - 1, dp);
        }
}


tree::~tree()
{
        for(int i = 0; i < this->nchildren; i++)
        {
                delete this->children[i];
        }

        delete this->tnp;
}


bool tree::copy(tree** to)
{
        //DEBUG_TREE_MSG("DEBUG: tree.cpp:");

        //init the 'to' tree with 'this's depth
        //(*to) = new tree(this->depth, this->dp);
        (*to) = (tree*)malloc(sizeof(class tree));
        (*to)->depth = this->depth;
```

```
        //copy tnp
        this->tnp->copy(&(*to)->tnp);

        //copy dp
        this->dp->copy(&(*to)->dp);

        //copy nchildren
        (*to)->nchildren = this->nchildren;

        //copy children
        for(int i = 0; i < this->nchildren; i++)
        {
                this->children[i]->copy(&(*to)->children[i]);
        }
        return(false);
}

tree_node *tree::gen_rand_nonterm_tree_node(darray *dp)
{
        /* initialize random seed: */
        srand ( clock() );

        /* generate secret number: */
        int type = rand() % NTYPES;

        DEBUGMSG("DEBUG: tree.cpp: Generating rand node with type " << type);
        switch (type)
        {
                case 0:
                {
                        this->tnp = new tree_node(tree_node::plus, 0.0, NULL);
                        break;
                }
                case 1:
                {
                        this->tnp = new tree_node(tree_node::minus, 0.0, NULL);
                        break;
                }
                case 2:
                {
                        this->tnp = new tree_node(tree_node::multi, 0.0, NULL);
                        break;
                }
                case 3:
                {
```

```
                        this->tnp = new tree_node(tree_node::div, 0.0, NULL);
                        break;
                }
                default:
                        cout << "DEBUG: tree.cpp: No type for node, got type" << typ
                        exit(1);
                }
}


tree_node *tree::gen_rand_term_tree_node(darray *dp)
{
        /* initialize random seed: */
        srand ( clock() );

        /* generate secret number: */
        int type = rand() % NTERMTYPES;

        DEBUGMSG("DEBUG: tree.cpp: Generating rand term node with type " << type);
        switch (type)
        {
                case 0:
                {
                        /* initialize random seed: */
                        srand ( clock() );

                        /* generate random double: */
                        double d = ((double)rand()/(double)RAND_MAX);

                        this->tnp = new tree_node(tree_node::tree_double, d, NULL);
                        break;
                }
                case 1:
                {
                        this->tnp = new tree_node(tree_node::tree_var, 0.0, &dp);
                        break;
                }
                default:
                        cout << "DEBUG: tree.cpp: No term type for node, got type "
                        exit(1);
                }
}


double tree::eval()
```

```
{
        switch(this->tnp->get_ntype())
        {
                //nonterminals
                case tree_node::plus:
                {
                        double sum = 0;
                        for(int i = 0; i < this->nchildren; i++)
                        {
                                sum += this->children[i]->eval();
                        }
                        return(sum);
                }
                case tree_node::minus:
                {
                        double sum = this->children[0]->eval();
                        for(int i = 1; i < this->nchildren; i++)
                        {
                                sum -= this->children[i]->eval();
                        }
                        return(sum);
                }
                case tree_node::multi:
                {
                        double prod = 1;
                        for(int i = 0; i < this->nchildren; i++)
                        {
                                prod *= this->children[i]->eval();
                        }
                        return(prod);
                }
                case tree_node::div:
                {
                        double quot = 1;
                        for(int i = 0; i < this->nchildren; i++)
                        {
                                //divide by zero safety
                                if(this->children[i]->eval() == 0)
                                {
                                        quot = 0;
                                }
                                else
                                {
                                        quot /= this->children[i]->eval();
                                }
```

```
                        }
                        return(quot);
                }

                //terminals
                case tree_node::tree_double:
                {
                        return(this->tnp->get_dval());
                }
                case tree_node::tree_var:
                {
                        return(this->tnp->get_ddp_val());
                }
                default:
                {
                        cerr << "ERROR: No type for eval()\n";
                        exit(1);
                }
        }
}


//set / change values in dp, and then run
double tree::fitness(double dexpected)
{
        return(abs(this->eval() - dexpected));
}


bool tree::is_term()
{
        if(this->nchildren <= 0)
        {
                return(true);
        }

        return(false);
}


bool tree::is_nonterm()
{
        if(this->nchildren <= 0)
        {
                return(false);
```

```
        }

        return(true);
}


int tree::count_terms()
{
        if(this->is_term() == true)
        {
                return(1);
        }

        int sum = 0;
        for(int i = 0; i < this->nchildren; i++)
        {
                sum += this->children[i]->count_terms();
        }

        return(sum);
}


int tree::count_nonterms()
{
        int sum = 0;

        if(this->is_nonterm() == true)
        {
                sum = 1;
        }

        for(int i = 0; i < this->nchildren; i++)
        {
                sum += this->children[i]->count_nonterms();
        }

        return(sum);
}


bool tree::crossover(tree **tp1, tree **tp2)
{
        if( (*tp1) == NULL || (*tp2) == NULL)
        {
```

```
                return(false);
        }

        int SUM_TEMP = 0;

        //Pick a random value of all nonterminals
        /* initialize random seed: */
        srand ( clock() );
        /* generate secret number: */
        int rand_val = rand() % (*tp1)->count_nonterms(); //0-n values
        DEBUG_TREE_MSG("DEBUG: tree.cpp: picking " << rand_val
                        << " nonterminal to "<< "crossover");

        //tree_crossover_nth_nonterm(tp1, tp2, rand_val);
}


bool tree::print(int depth)
{
        if(this == NULL)
        {
                //false if I am a child that didn't get a value
                return(false);
        }

        cout << string(depth, ' ') << depth << ":";
        this->tnp->print_ntype();
        cout << " = " << this->eval();
        //more debugging stuff
        //cout << ", term:nonterm == " << this->is_term() << ":" << this->is_nonterm
        //cout << " nterm:nnonterm == " << this->count_terms() << ":" << this->count
        cout << ", children = " << this->nchildren;
        cout << endl;

        for(int i = 0; i < this->nchildren; i++)
        {
                this->children[i]->print(depth + 1);
        }

        return(true);
}


bool tree::print_tnp_ntype()
{
```

```
        if(this == NULL)
        {
                return(false);
        }
        return(this->tnp->print_ntype());
}


////////////////////////////////////////////////////////////////////////////
//External tree functions
////////////////////////////////////////////////////////////////////////////

bool mutate_nth_nonterm(tree **tp, int n, int depth, int new_depth, darray *dp)
{
        if( (*tp) == NULL)
        {
                return(false);
        }

        if( (*tp)->is_nonterm())
        {
                SUM_TEMP++;
        }
        cout << string(depth, ' ') << depth << ":";
        (*tp)->print_tnp_ntype();
        cout << " = " << SUM_TEMP;

        if(n == SUM_TEMP && (*tp)->is_nonterm())
        {
                cout << " !mutating!";

                //set this tree node to a new rand tree until it is a
                // nonterminal
                do
                {
                        delete (*tp);
                        (*tp) = new tree(new_depth, dp);
                } while ((*tp)->is_nonterm() != true);

                cout << endl;
                return(true);
        }

        cout << endl;
```

```
        //if we've already see the node to mutate

        if(n < SUM_TEMP)
        {
                return(true);
        }

        for(int i = 0; i < (*tp)->nchildren; i++)
        {
                mutate_nth_nonterm(&(*tp)->children[i], n, depth + 1, new_depth,
                                                        dp);
        }

        return(true);
}


bool tree_replace_nth_nonterm(tree **tp, tree **with, int n)
{
        if((*tp) == NULL)
        {
                return(false);
        }

        if((*tp)->is_nonterm())
        {
                SUM_TEMP++;

                if(n == SUM_TEMP)
                {
                        //copy
                        delete (*tp);
                        (*with)->copy(&(*tp));
                        return(true);
                }
                else
                {
                        for(int i = 0; i < (*tp)->nchildren; i++)
                        {
                                bool status = \
                                 tree_replace_nth_nonterm(
                                                        &(*tp)->children[i],
                                                        &(*with),
                                                        n
                                                        );
```

```
                                if(status == true) {return(true);}
                        }
                }
        }

        return(false);
}


bool tree_crossover(tree **tp1, tree **tp2)
{
        //make temp; tp1 original to crossover with tp2
        tree *tp1_temp;
        (*tp1)->copy(&tp1_temp);


        //crossover - tp1
        //int rand_val;
        // create random nonterm index
        /* initialize random seed: */
        srand ( clock() );
        /* generate secret number: */
        int rand_val = rand() % (*tp1)->count_nonterms(); //0-n values
        // replace
        DEBUG_TREE_MSG( "tree.cpp: tree 1 crossover on " << rand_val);
        #ifdef DEBUG_TREE
        cout << " out of " << (*tp1)->count_nonterms() << endl;
        #endif
        tree_replace_nth_nonterm(&(*tp1), &(*tp2), rand_val);

        //crossover - tp2
        // create random nonterm index
        /* generate secret number: */
        rand_val = rand() % (*tp2)->count_nonterms(); //0-n values
        DEBUG_TREE_MSG( "tree.cpp: tree 2 crossover on " << rand_val);
        #ifdef DEBUG_TREE
        cout << " out of " << (*tp2)->count_nonterms() << endl;
        #endif
        // replace
        tree_replace_nth_nonterm(&(*tp2), &tp1_temp, rand_val);

}
```

9.8 test.h

```
#ifndef _TEST_H
```

```
#define _TEST_H

bool test_nodes();
bool test_darray();
bool test_trees();
bool test_tree_copy();
bool test_tree_replace();
bool test_tree_crossover();

#endif
```

9.9 test.cpp

```
#include <iostream>
#include <stdlib.h>
#include "main.h"
#include "tree_node.h"
#include "tree.h"


bool test_nodes()
{
        //create nodes
        darray *dp = new darray(200, true);
        tree_node *tp;
        tp = new tree_node(tree_node::plus, 0.0, NULL);
        delete tp;
        tp = new tree_node(tree_node::minus, 0.0, NULL);
        delete tp;
        tp = new tree_node(tree_node::multi, 0.0, NULL);
        delete tp;
        tp = new tree_node(tree_node::div, 0.0, NULL);
        delete tp;
        tp = new tree_node(tree_node::tree_double, 2.001, NULL);
        delete tp;
        tp = new tree_node(tree_node::tree_var, 0.0, &dp);
        delete tp;
        delete dp;

        //copy test
        cout << "Tree node copy test\n";
        cout << " Plus:\n";
        darray *dp1 = new darray(10, true);
        tree_node *tnp1;
        tree_node *tnp2;
        tnp1 = new tree_node(tree_node::plus, 0.0, NULL);
```

```
        tnp1->copy(&tnp2);
        tnp1->print_members();
        delete tnp1;
        tnp2->print_members();
        delete tnp2;
        delete dp1;

        cout << " Tree var:\n";
        dp1 = new darray(2, false);
        dp1->a[0] = 5;
        dp1->a[1] = 7;
        cout << "TS45: " << dp1 << endl;
        tnp1 = new tree_node(tree_node::tree_var, 0.0, &dp1);
        tnp1->copy(&tnp2);
        tnp1->print_members();
        delete tnp1;
        tnp2->print_members();
        dp1->a[0] = 9;
        dp1->a[1] = 9;
        tnp2->print_members();

        delete dp1;
}


bool test_darray()
{
        darray *dp1 = new darray(5, true);
        darray *dp2;
        dp1->copy(&dp2);

        cout << "test_darray: dp1: \n";
        dp1->print_vals();
        delete dp1;
        cout << "test_darray: dp2: \n";
        dp2->print_vals();
        delete dp2;
}


bool test_trees()
{
        tree *tp;
        darray *dp = new darray(200, true);
        dp->a[0] = 0.2;
```

```
dp->a[1] = 0.3;

//test making lots of trees
for(int i = 0; i < 500; i++)
{
        tp = new tree(5, dp);
        delete tp;
}
delete dp;
cout << "Finished bulk tree creation test\n";

//eval a tree
cout << "Eval tree test\n";
dp = new darray(2, false);
dp->a[0] = 0.2;
dp->a[1] = 0.3;
tp = new tree(5, dp);
tp->print(0);
cout << "Tree has " << tp->count_terms() << " terminal(s).\n";
cout << "Tree has " << tp->count_nonterms() << " non-terminal(s).\n";
delete dp;
delete tp;

//deep tree eval
cout << "Deep tree eval time test: ";
clock_t stime, etime, ttime;
int precision = 1000;
stime = (clock () / CLOCKS_PER_SEC) * precision;
dp = new darray(2, false);
dp->a[0] = 0.2;
dp->a[1] = 0.3;
tp = new tree(16, dp);
etime = (clock () / CLOCKS_PER_SEC) * precision;
ttime = (etime - stime) / precision;
cout << ttime << " second(s)\n";
//tp->print(0);
delete dp;
delete tp;

//Mutate test
tp = new tree(5, dp);
dp = new darray(2, false);
dp->a[0] = 0.2;
dp->a[1] = 0.3;
int n = 10;
```

```
        cout << "Term mutation on " << n << " terminal\n";
        SUM_TEMP = 0;
        mutate_nth_nonterm(&tp, n, 0, 5, dp);
        cout << "After mutation:\n";
        tp->print(0);

        //x^3 + 5y^3 - 4xy + 7
        //= (.2)^3 + 5(.3)^3 - 4(.2)(.3) + 7
        //= .008 + .135 - .24 + 7
        //= 6.903
        dp->a[0] = 0.2;
        dp->a[1] = 0.3;
        cout << "Tree fitness: " << tp->fitness(6.903) << endl;
        cout << "Tree eval 1: " << tp->eval() << endl;

        //x^3 + 5y^3 - 4xy + 7
        //
        dp->a[0] = 5;
        dp->a[1] = 7;
        cout << "Tree eval 2: " << tp->eval() << endl;

        //x^3 + 5y^3 - 4xy + 7
        //
        dp->a[0] = 13;
        dp->a[1] = 20;
        cout << "Tree eval 3: " << tp->eval() << endl;

        delete dp;
}


bool test_tree_copy()
{
        //Crossover test
        darray *dp1 = new darray(2, false);
        darray *dp2 = new darray(2, false);
        dp1->a[0] = 0.2;
        dp1->a[1] = 0.3;
        dp2->a[0] = 5;
        dp2->a[1] = 7;
        tree *tp1 = new tree(5, dp1);
        tree *tp2 = NULL;

        tp1->copy(&tp2);
        cout << "Tree copy test\n";
```

```
        cout << " Tree 1:\n";
        //tp1->print(0);
        cout << " " << tp1->eval() << endl;
        delete tp1;
        cout << " Tree 2:\n";
        //tp2->print(0);
        cout << " " << tp2->eval() << endl;

        delete tp2;
        delete dp1;
        delete dp2;
}


bool test_tree_replace()
{
        darray *dp1 = new darray(2, false);
        dp1->a[0] = 0.2;
        dp1->a[1] = 0.3;

        tree *tp1 = new tree(5, dp1);
        tree *tp2 = new tree(5, dp1);

        cout << "Tree replace test\n";
        cout << "Tree 2:\n";
        tp2->print(0);

        cout << "Tree 1 before replace:\n";
        tp1->print(0);
        SUM_TEMP = 0;
        tree_replace_nth_nonterm(&tp1, &tp2, 4);
        delete tp2;
        cout << "Tree 1 after replace:\n";
        tp1->print(0);


        delete tp1;
}

bool test_tree_crossover()
{
        darray *dp1 = new darray(2, false);
        dp1->a[0] = 0.2;
        dp1->a[1] = 0.3;
```

```
        tree *tp1 = new tree(5, dp1);
        tree *tp2 = new tree(5, dp1);

        cout << "Tree crossover test:\n";
        tree_crossover(&tp1, &tp2);
        cout << "Tree 1 after crossover:\n";
        tp1->print(0);
        cout << "Tree 2 after crossover:\n";
        tp2->print(0);
}
```