Assignment #HW1Alt – Genetic Programming

CSSE490: Bio-Inspired Artificial Intelligence

Prepared and submitted by: Edward Kim

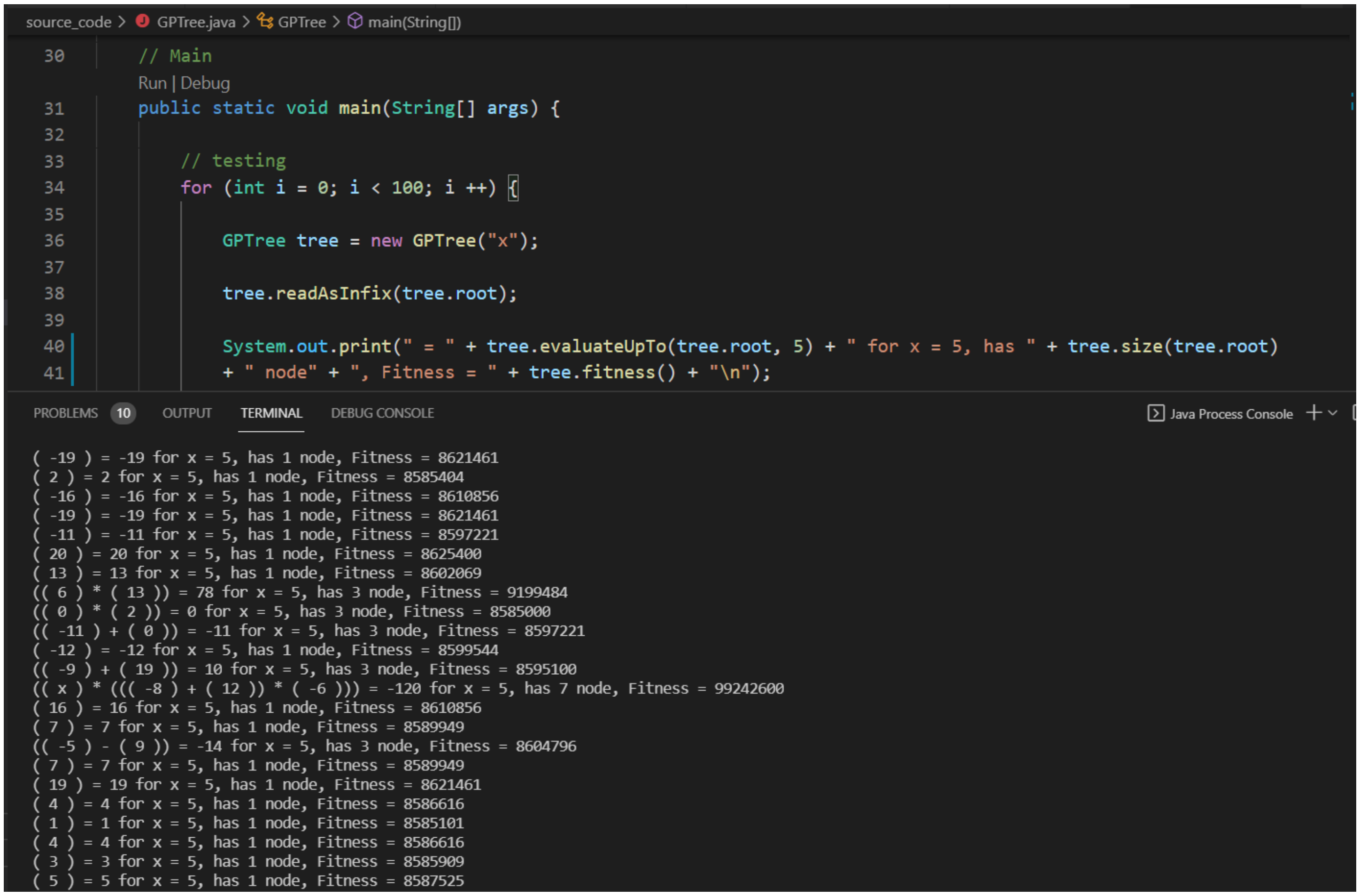
Collaboration and resources: I worked alone.

Resources I used to complete this assignment (websites, textbook, friends, etc.):

Java official documentation

Edward Kim's genetic algorithm project source code from 2021W CSSE220

Checkpoint 1 – Tree-based Data Structure / Checkpoint 2 – Expression Evaluator / Checkpoint 3 – Fitness Function / Checkpoint 4 – Generating Random Expression Tree

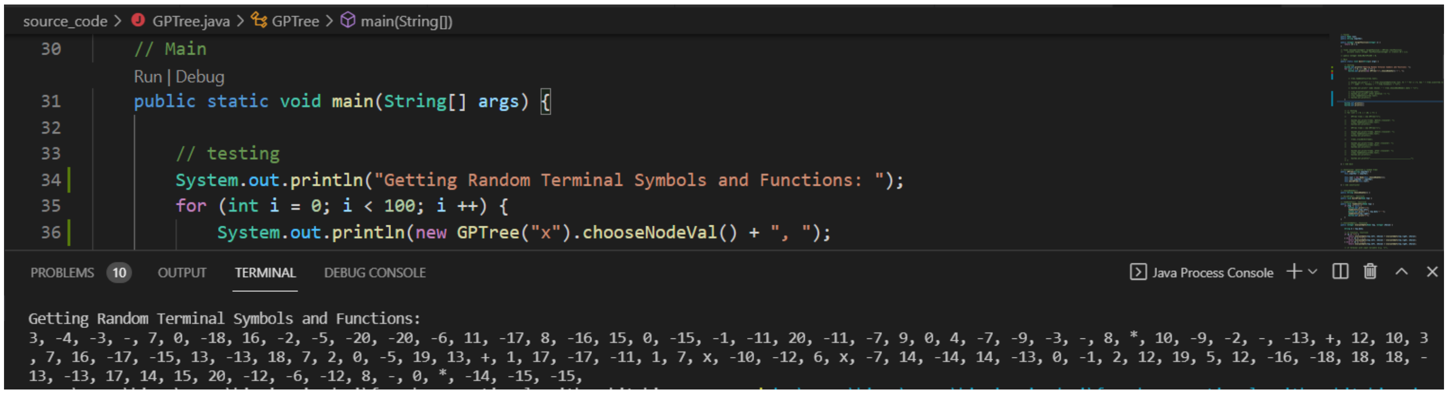


```java
    // Main
    public static void main(String[] args) {

        // testing
        for (int i = 0; i < 100; i ++) {

            GPTree tree = new GPTree("x");

            tree.readAsInfix(tree.root);

            System.out.print(" = " + tree.evaluateUpTo(tree.root, 5) + " for x = 5, has " + tree.size(tree.root)
            + " node" + ", Fitness = " + tree.fitness() + "\n");
```

```
( -19 ) = -19 for x = 5, has 1 node, Fitness = 8621461
( 2 ) = 2 for x = 5, has 1 node, Fitness = 8585404
( -16 ) = -16 for x = 5, has 1 node, Fitness = 8610856
( -19 ) = -19 for x = 5, has 1 node, Fitness = 8621461
( -11 ) = -11 for x = 5, has 1 node, Fitness = 8597221
( 20 ) = 20 for x = 5, has 1 node, Fitness = 8625400
( 13 ) = 13 for x = 5, has 1 node, Fitness = 8602069
(( 6 ) * ( 13 )) = 78 for x = 5, has 3 node, Fitness = 9199484
(( 0 ) * ( 2 )) = 0 for x = 5, has 3 node, Fitness = 8585000
(( -11 ) + ( 0 )) = -11 for x = 5, has 3 node, Fitness = 8597221
( -12 ) = -12 for x = 5, has 1 node, Fitness = 8599544
(( -9 ) + ( 19 )) = 10 for x = 5, has 3 node, Fitness = 8595100
(( x ) * ((( -8 ) + ( 12 )) * ( -6 ))) = -120 for x = 5, has 7 node, Fitness = 99242600
( 16 ) = 16 for x = 5, has 1 node, Fitness = 8610856
( 7 ) = 7 for x = 5, has 1 node, Fitness = 8589949
(( -5 ) - ( 9 )) = -14 for x = 5, has 3 node, Fitness = 8604796
( 7 ) = 7 for x = 5, has 1 node, Fitness = 8589949
( 19 ) = 19 for x = 5, has 1 node, Fitness = 8621461
( 4 ) = 4 for x = 5, has 1 node, Fitness = 8586616
( 1 ) = 1 for x = 5, has 1 node, Fitness = 8585101
( 4 ) = 4 for x = 5, has 1 node, Fitness = 8586616
( 3 ) = 3 for x = 5, has 1 node, Fitness = 8585909
( 5 ) = 5 for x = 5, has 1 node, Fitness = 8587525
```

The above screenshot demonstrates: 1) generating a random binary expression tree, 2) representing it in infix expression, 3) evaluating the value of the expression tree, assuming an arbitrary value for "x" (5 in the above case), 4) counting the size (number of nodes) of the tree, 5) and evaluating its fitness.

Checkpoint 4 – Getting Random Terminal Symbols and Functions



source_code > GPTree.java > GPTree > main(String[])

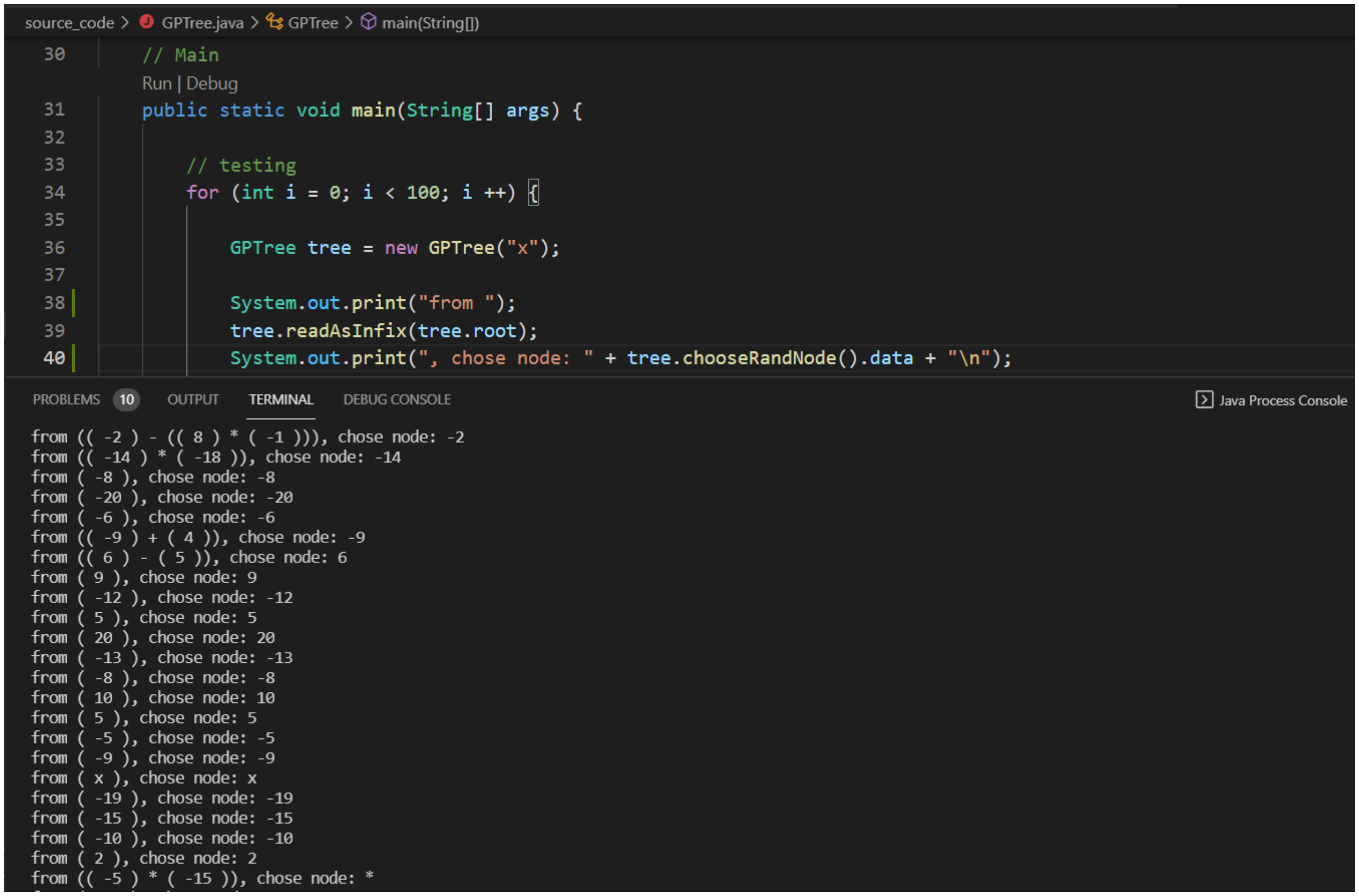
```java
    // Main
    Run | Debug
    public static void main(String[] args) {

        // testing
        System.out.println("Getting Random Terminal Symbols and Functions: ");
        for (int i = 0; i < 100; i ++) {
            System.out.println(new GPTree("x").chooseNodeVal() + ", ");
```

PROBLEMS 10 OUTPUT TERMINAL DEBUG CONSOLE Java Process Console

```
Getting Random Terminal Symbols and Functions:
3, -4, -3, -, 7, 0, -18, 16, -2, -5, -20, -20, -6, 11, -17, 8, -16, 15, 0, -15, -1, -11, 20, -11, -7, 9, 0, 4, -7, -9, -3, -, 8, *, 10, -9, -2, -, -13, +, 12, 10, 3
, 7, 16, -17, -15, 13, -13, 18, 7, 2, 0, -5, 19, 13, +, 1, 17, -17, -11, 1, 7, x, -10, -12, 6, x, -7, 14, -14, 14, -13, 0, -1, 2, 12, 19, 5, 12, -16, -18, 18, 18, -
13, -13, 17, 14, 15, 20, -12, -6, -12, 8, -, 0, *, -14, -15, -15,
```

This random node value generation has also been used in the first screenshot to generate random expression trees.

Checkpoint 4 – Choosing Random Node in Tree



source_code > GPTree.java > GPTree > main(String[])

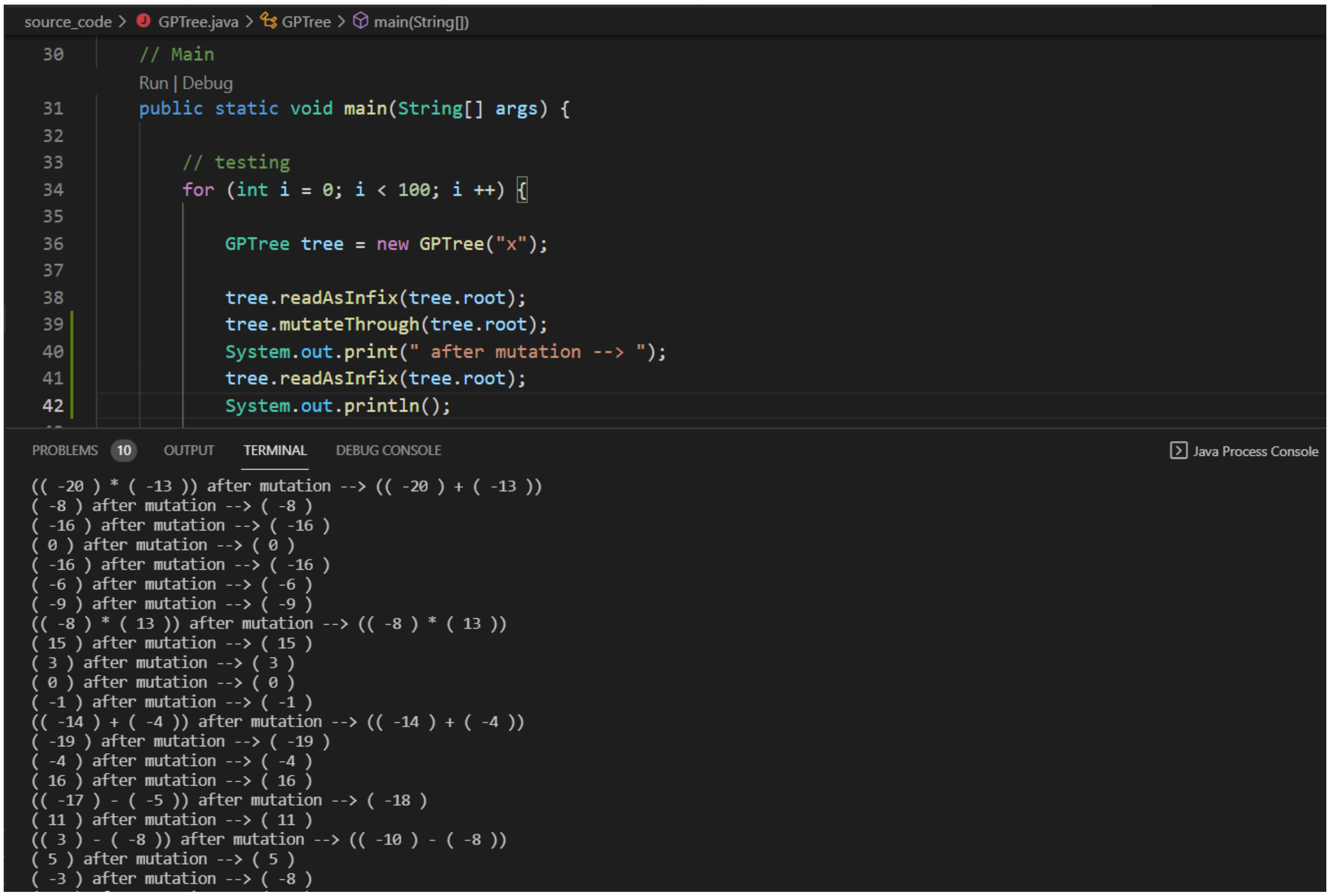
```java
    // Main
    Run | Debug
    public static void main(String[] args) {

        // testing
        for (int i = 0; i < 100; i ++) {

            GPTree tree = new GPTree("x");

            System.out.print("from ");
            tree.readAsInfix(tree.root);
            System.out.print(", chose node: " + tree.chooseRandNode().data + "\n");
```

PROBLEMS 10 OUTPUT TERMINAL DEBUG CONSOLE Java Process Console

```
from (( -2 ) - (( 8 ) * ( -1 ))), chose node: -2
from (( -14 ) * ( -18 )), chose node: -14
from ( -8 ), chose node: -8
from ( -20 ), chose node: -20
from ( -6 ), chose node: -6
from (( -9 ) + ( 4 )), chose node: -9
from (( 6 ) - ( 5 )), chose node: 6
from ( 9 ), chose node: 9
from ( -12 ), chose node: -12
from ( 5 ), chose node: 5
from ( 20 ), chose node: 20
from ( -13 ), chose node: -13
from ( -8 ), chose node: -8
from ( 10 ), chose node: 10
from ( 5 ), chose node: 5
from ( -5 ), chose node: -5
from ( -9 ), chose node: -9
from ( x ), chose node: x
from ( -19 ), chose node: -19
from ( -15 ), chose node: -15
from ( -10 ), chose node: -10
from ( 2 ), chose node: 2
from (( -5 ) * ( -15 )), chose node: *
```

Choosing random node for use in crossover.

Checkpoint 5 – Mutation

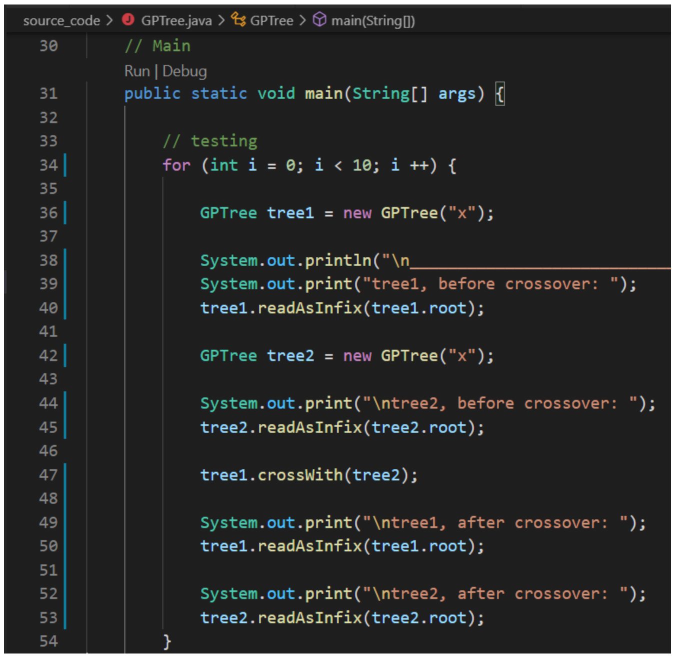


```java
    // Main
    public static void main(String[] args) {

        // testing
        for (int i = 0; i < 100; i ++) {

            GPTree tree = new GPTree("x");

            tree.readAsInfix(tree.root);
            tree.mutateThrough(tree.root);
            System.out.print(" after mutation --> ");
            tree.readAsInfix(tree.root);
            System.out.println();
```

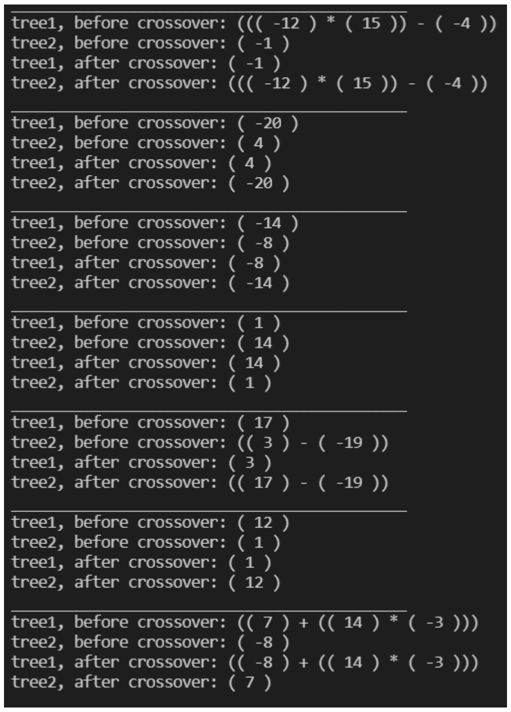
```
(( -20 ) * ( -13 )) after mutation --> (( -20 ) + ( -13 ))
( -8 ) after mutation --> ( -8 )
( -16 ) after mutation --> ( -16 )
( 0 ) after mutation --> ( 0 )
( -16 ) after mutation --> ( -16 )
( -6 ) after mutation --> ( -6 )
( -9 ) after mutation --> ( -9 )
(( -8 ) * ( 13 )) after mutation --> (( -8 ) * ( 13 ))
( 15 ) after mutation --> ( 15 )
( 3 ) after mutation --> ( 3 )
( 0 ) after mutation --> ( 0 )
( -1 ) after mutation --> ( -1 )
(( -14 ) + ( -4 )) after mutation --> (( -14 ) + ( -4 ))
( -19 ) after mutation --> ( -19 )
( -4 ) after mutation --> ( -4 )
( 16 ) after mutation --> ( 16 )
(( -17 ) - ( -5 )) after mutation --> ( -18 )
( 11 ) after mutation --> ( 11 )
(( 3 ) - ( -8 )) after mutation --> (( -10 ) - ( -8 ))
( 5 ) after mutation --> ( 5 )
( -3 ) after mutation --> ( -8 )
```

Architecture-altering mutations are also implemented, changing a terminal node to a function node (and randomly building the new subtree) or changing a function node to a terminal node (and removing the previous child nodes).

Checkpoint 5 – Crossover

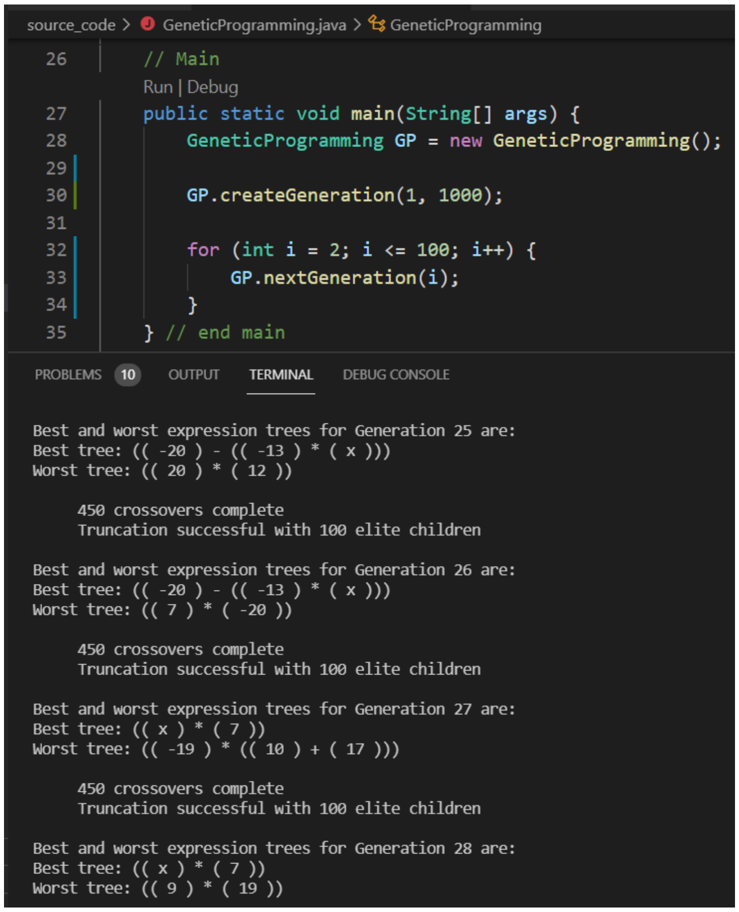


```java
    // Main
    public static void main(String[] args) {

        // testing
        for (int i = 0; i < 10; i ++) {

            GPTree tree1 = new GPTree("x");

            System.out.println("\n________________________________
            System.out.print("tree1, before crossover: ");
            tree1.readAsInfix(tree1.root);

            GPTree tree2 = new GPTree("x");

            System.out.print("\ntree2, before crossover: ");
            tree2.readAsInfix(tree2.root);

            tree1.crossWith(tree2);

            System.out.print("\ntree1, after crossover: ");
            tree1.readAsInfix(tree1.root);

            System.out.print("\ntree2, after crossover: ");
            tree2.readAsInfix(tree2.root);
        }
```



```
tree1, before crossover: ((( -12 ) * ( 15 )) - ( -4 ))
tree2, before crossover: ( -1 )
tree1, after crossover: ( -1 )
tree2, after crossover: ((( -12 ) * ( 15 )) - ( -4 ))
________________________________
tree1, before crossover: ( -20 )
tree2, before crossover: ( 4 )
tree1, after crossover: ( 4 )
tree2, after crossover: ( -20 )
________________________________
tree1, before crossover: ( -14 )
tree2, before crossover: ( -8 )
tree1, after crossover: ( -8 )
tree2, after crossover: ( -14 )
________________________________
tree1, before crossover: ( 1 )
tree2, before crossover: ( 14 )
tree1, after crossover: ( 14 )
tree2, after crossover: ( 1 )
________________________________
tree1, before crossover: ( 17 )
tree2, before crossover: (( 3 ) - ( -19 ))
tree1, after crossover: ( 3 )
tree2, after crossover: (( 17 ) - ( -19 ))
________________________________
tree1, before crossover: ( 12 )
tree2, before crossover: ( 1 )
tree1, after crossover: ( 1 )
tree2, after crossover: ( 12 )
________________________________
tree1, before crossover: (( 7 ) + (( 14 ) * ( -3 )))
tree2, before crossover: ( -8 )
tree1, after crossover: (( -8 ) + (( 14 ) * ( -3 )))
tree2, after crossover: ( 7 )
```

Checkpoint 6 – Evolutionary Loop


source_code > GeneticProgramming.java > GeneticProgramming

```java
    // Main
    Run | Debug
    public static void main(String[] args) {
        GeneticProgramming GP = new GeneticProgramming();

        GP.createGeneration(1, 1000);

        for (int i = 2; i <= 100; i++) {
            GP.nextGeneration(i);
        }
    } // end main
```

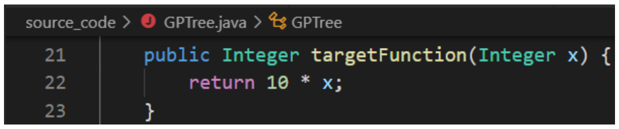
```
Best and worst expression trees for Generation 25 are:
Best tree: (( -20 ) - (( -13 ) * ( x )))
Worst tree: (( 20 ) * ( 12 ))

     450 crossovers complete
     Truncation successful with 100 elite children

Best and worst expression trees for Generation 26 are:
Best tree: (( -20 ) - (( -13 ) * ( x )))
Worst tree: (( 7 ) * ( -20 ))

     450 crossovers complete
     Truncation successful with 100 elite children

Best and worst expression trees for Generation 27 are:
Best tree: (( x ) * ( 7 ))
Worst tree: (( -19 ) * (( 10 ) + ( 17 )))

     450 crossovers complete
     Truncation successful with 100 elite children

Best and worst expression trees for Generation 28 are:
Best tree: (( x ) * ( 7 ))
Worst tree: (( 9 ) * ( 19 ))
```

Checkpoint 7 – Experiments

See next page.

Experiment 1: Validation of Simplest Example

F(x) = x * 10



```java
public Integer targetFunction(Integer x) {
    return 10 * x;
}
```

Setup:

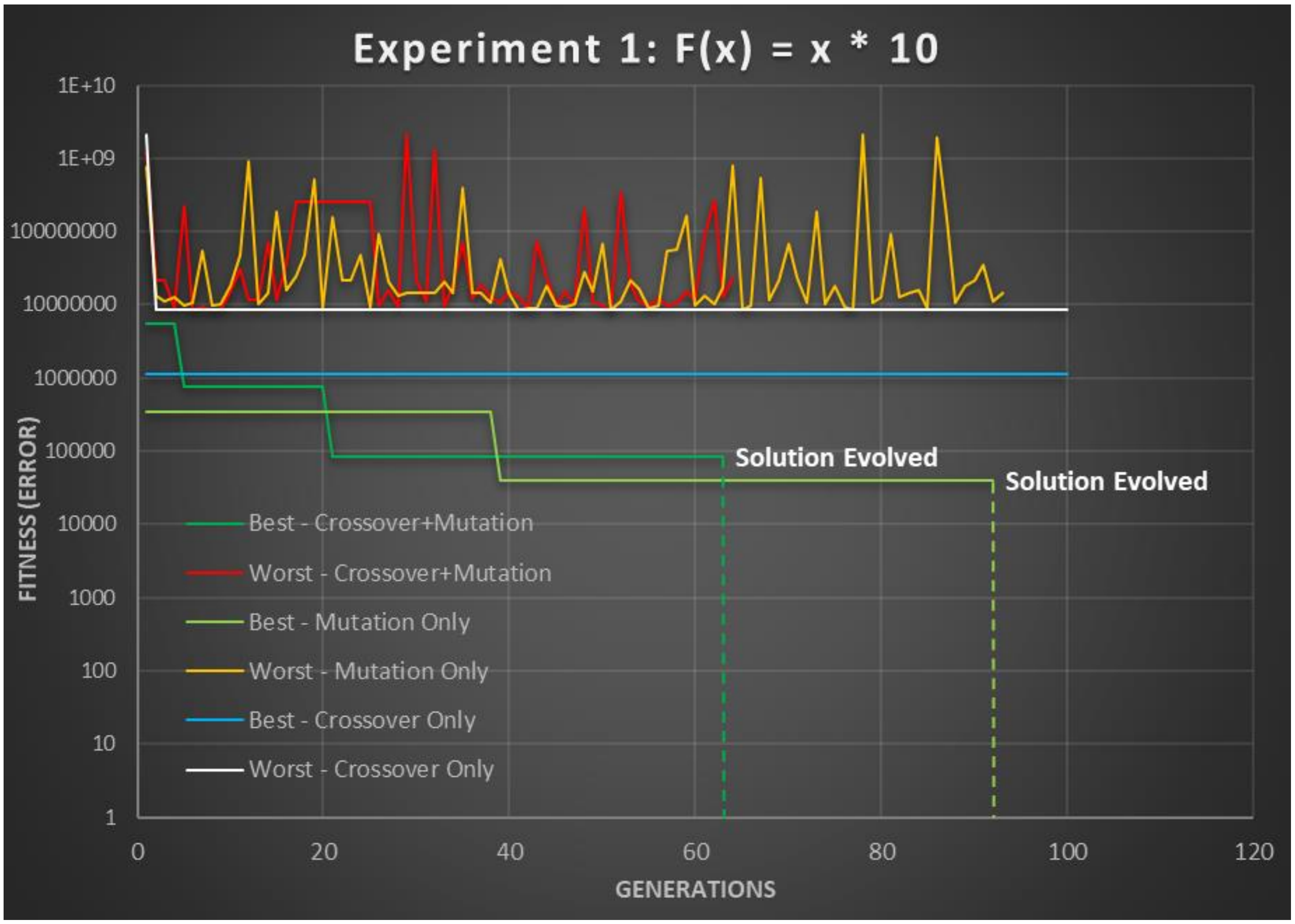
Parameter	Trial 1 (crossover + mutation)	Trial 2 (mutation only)	Trial 3 (crossover only)
Mutation	On	On	Off
Crossover	On	Off	Off
Mutate Probability (per node)	10%	10%	10%
Crossover Point	Random	Random	Random
Elitism (%)	10	10	10
Max Generations	100	100	100
Population Size	1000	1000	1000
Selection Method	Truncation	Truncation	Truncation
Termination Fitness	10	10	10

Besides the given options for crossover and/or mutation in each trial, elitism was implemented and activated for all three trials as it was found to significantly improve the evolutionary progress; preliminary experiments showed near-perfect solution trees simply being lost after one or two generations due to mutation.

Hypothesis:

The diversity of the tree population, and hence its evolutionary success, will increase significantly from Trial 3 to Trial 2, and again from Trial 2 to Trial 1, as the constraints on Trial 3 and Trial 2 make their reachable search space much more dependent on the randomly generated starting population compared to that of Trial 1 (and similarly so when comparing Trial 3 to Trial 2).

Results and Conclusion:



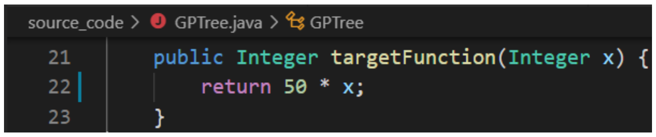
It can first be seen that for all three trials, the best fitness (lowest error) never increases due to elitism, whereas the worst fitness (highest error) fluctuates across generations with the exception of Trial 3 (crossover only). The worst fitness in Trial 3 decreases at generation 2 (see Data_Genetic_Programming.xlsx in the same folder), but remains constant until the termination at generation 100. The best fitness in Trial 3 stays constant throughout the whole duration of the evolution, indicating that it was not able to evolve a better tree than the best tree generated in the starting population ("(((-11) - (x)) * (-8))"). This can be attributed to the constraint placed on Trial 3 by disabling mutation, as the genetic material available in the population is always limited to what was created at generation 1; the evolution can only attempt to recombine and restructure the trees with the same material.

Besides, it is also shown that Trial 1 and Trial 2 have successfully evolved a solution, which were "((x) * (10))" and "((((-8) * (-9)) * (0)) + ((x) * (10)))", respectively. Their difference, despite being algebraically equivalent, exemplifies the open-ended nature of evolution. A non-evolutionary approach would have easily identified that multiplying 0 as in Trial 2's solution neutralizes the product, but since evolution does not know such principle, it is in turn able to search for and arrive at a wider variety of solutions for any given objective — which, in some cases, may even be completely novel compared to a non-evolutionary approach (such as conventional design).

Trial 1 evolved the solution in 64 generations, and Trial 2 evolved the solution in 93 generations (just before termination). The relative success of Trial 1 is not surprising as it combines both crossover and mutation to enhance the population's diversity, but it is still interesting to examine the evolutionary trajectory in both cases. Extracting the transition points where a new best tree was generated, Trial 1 went from "(x)" in generation 1, "((x) * (7))" in generation 5, "((9) * (x))" in generation 21, to finally "((x) * (10))" in generation 64. In the case of Trial 2, the new best tree alternated between “((x) * (12))” and “((8) * (x))” from generations 1 to 38, then went to “((10) * ((-2) + (x)))” in generation 39, terminating with “((((-8) * (-9)) * (0)) + ((x) * (10)))” in generation 93. These are very interesting behaviors: gradually reaching a form closer to the expected solution (e.g. “((x) * (7))” and “((9) * (x))” in Trial 1, and especially the near-perfect solution “((10) * ((-2) + (x)))” in Trial 2), competition between two equally fit individuals for an extended period of time (“((x) * (12))” and “((8) * (x))” in Trial 2), and the termination of such competition by a more fit newcomer (“((10) * ((-2) + (x)))”).

Experiment 2: Finding a solution that requires multiple terminal numbers

F(x) = x * 50



```java
    public Integer targetFunction(Integer x) {
        return 50 * x;
    }
```

Setup:

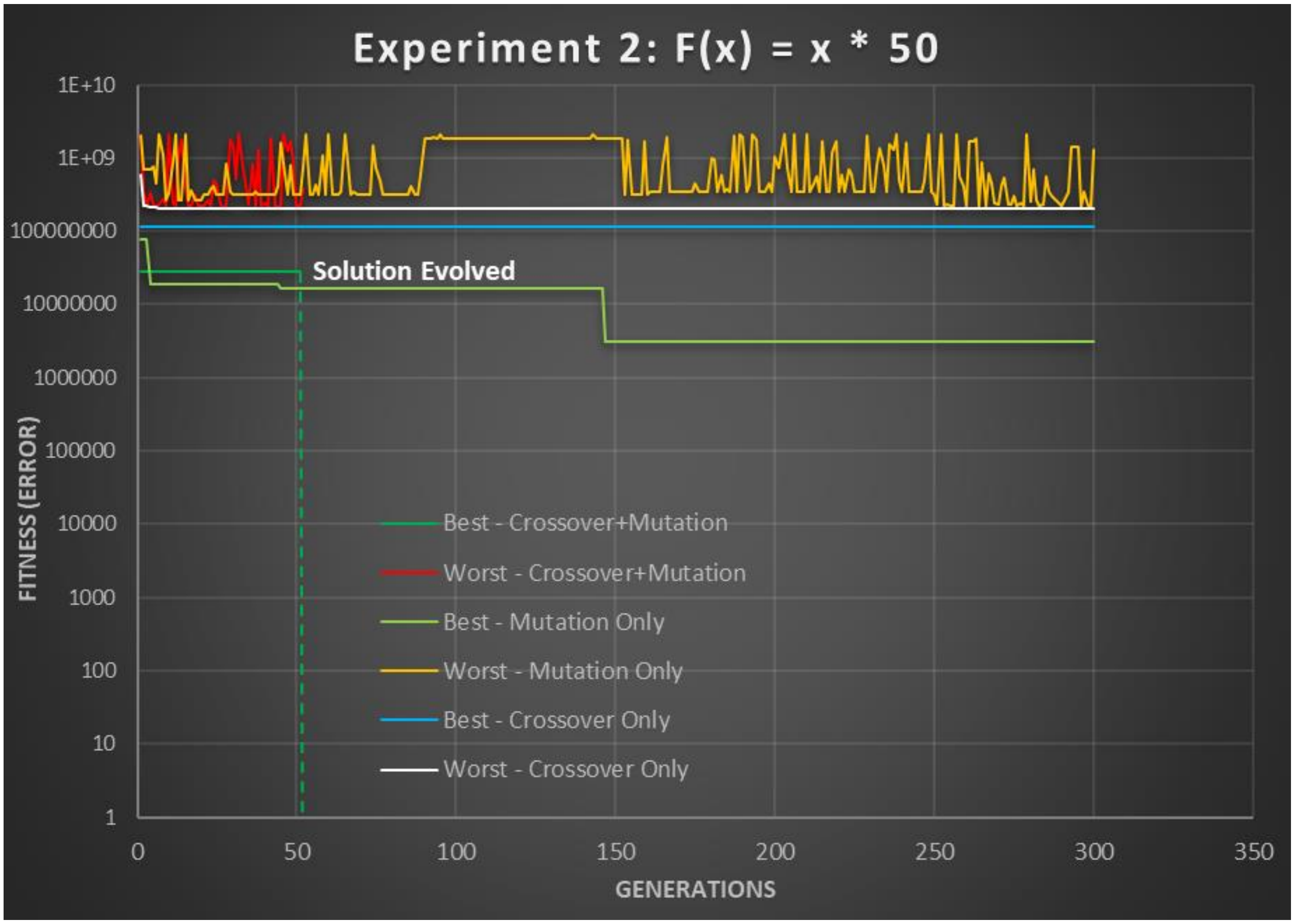
Parameter	Trial 1 (crossover + mutation)	Trial 2 (mutation only)	Trial 3 (crossover only)
Mutation	On	On	Off
Crossover	On	Off	Off
Mutate Probability (per node)	10%	10%	10%
Crossover Point	Random	Random	Random
Elitism (%)	1	1	1
Max Generations	300	300	300
Population Size	3000	3000	3000
Selection Method	Truncation	Truncation	Truncation
Termination Fitness	10	10	10

This experiment intends to explore evolving towards an expression that cannot be directly encoded in the tree, due to the limit of terminal nodes having a value in range [-20, 20]. As such, it extends on from the aforementioned, open-ended nature of evolution, as it would show what kind of creative paths evolution would take to reach a tree that is algebraically equivalent to x * 50. Preliminary experiments have shown the configuration with 1% elitism, 300 generations, and population size 3000 to be one of the most time-efficient in evolving a decent solution to this problem, and hence was applied to all three trials.

Hypothesis:

Compared to Experiment 1, all three trials will be a lot more difficult to evolve a solution (take more generations) as the definitive solution to the problem ("(x) * (50))") cannot exist in the search space — the problem does not have a clear epicenter around which a high concentration of similar, near-perfect solutions exist. As such, the solutions successfully evolved (if any) will also differ significantly in their architecture.

Results and conclusion:



It can be seen again that for all three trials, the best fitness (lowest error) never increases due to elitism, whereas the worst fitness (highest error) fluctuates across generations, with the exception of Trial 3 (crossover only) due to the limited genetic material available in the population. Trial 3, again, was not able to evolve a better tree than the best tree generated in the starting population ("((x) * (13))"), shown above in the constant best fitness line for Trial 3.

Trial 1 was able to evolve a solution ("((-10) * ((-5) * (x)))") surprisingly quickly in just 52 generations, but it is to be noted that this was a very rare success most likely thanks to luck, with good genetic material randomly generated at the beginning and throughout the generations. Yet, this exemplifies an important aspect of evolution, that it is affected by luck quite significantly. If an evolutionary step happens to proceed in a direction that is highly advantageous, the succeeding evolution can reach the given objective abnormally quickly or accurately. In the case of Trial 1, the best tree from the very beginning (generation 1) has been "(((10) - ((-8) + (-14))) * (x))", which is already quite close to the final solution as it is algebraically equivalent to "((32) * (x))".

With Trial 2, mutations alone were not able to successfully evolve a solution, but they reached "((7) * ((8) * (x)))" by generation 147, algebraically equivalent to "((56) * (x))". Such observation that mutation alone can get to an approximate solution in a relatively short time, but may require more time to evolve the solution, is repeated from Experiment 1 and can again be attributed to the relative lack of diversity due to the absence of crossover.