Lab 2
Gene Expression

```python
import random
import math

# Target function to approximate
def target_function(x):
    return x**2 + x + 1

# Terminal and function sets
TERMINALS = ['x']
FUNCTIONS = ['+', '-', '*', '/']

# Parameters
POP_SIZE = 100
MUTATION_RATE = 0.1
TOURNAMENT_SIZE = 3
GENERATIONS = 100

# ---------------------
# Utility functions
# ---------------------

def safe_div(a, b):
    """Safe division to avoid divide-by-zero."""
    try:
        return a / b if b != 0 else 1
    except:
        return 1

def random_gene(max_depth=3):
    """Randomly create an expression (gene) of limited depth."""
    if max_depth == 0 or (max_depth > 1 and random.random() < 0.3):
        return random.choice(TERMINALS)
    else:
        op = random.choice(FUNCTIONS)
        left = random_gene(max_depth - 1)
        right = random_gene(max_depth - 1)
        return f"({left}{op}{right})"

def evaluate_expression(expr, x):
    """Safely evaluate the evolved expression."""
    try:
```

```python
def random_gene(max_depth=3):
    """Randomly create an expression (gene) of limited depth."""
    if max_depth == 0 or (max_depth > 1 and random.random() < 0.3):
        return random.choice(TERMINALS)
    else:
        op = random.choice(FUNCTIONS)
        left = random_gene(max_depth - 1)
        right = random_gene(max_depth - 1)
        return f"({left}{op}{right})"

def evaluate_expression(expr, x):
    """Safely evaluate the evolved expression."""
    try:
        # Define the environment so eval can use math and safe_div properly
        env = {"x": x, "math": math, "safe_div": safe_div}
        # Replace '/' with safe_div(a,b)
        expr_safe = expr.replace("/", "safe_div")
        result = eval(expr_safe, env)
        if callable(result):  # in case a function is accidentally returned
            return 0
        return float(result)
    except Exception:
        return 0

def fitness(expr):
    """Mean squared error against target function."""
    xs = [i for i in range(-10, 11)]
    errors = []
    for x in xs:
        y_true = target_function(x)
        y_pred = evaluate_expression(expr, x)
        errors.append((y_true - y_pred)**2)
    mse = sum(errors) / len(errors)
    return mse

# ---------------------
# Genetic operators
# ---------------------

def mutate(expr):
    """Randomly mutate part of the expression."""
    expr_list = list(expr)
    for i in range(len(expr_list)):
        if random.random() < MUTATION_RATE:
            expr_list[i] = random.choice(['x', '+', '-', '*', '/'])
    return ''.join(expr_list)
```

```python
def mutate(expr):
    """Randomly mutate part of the expression."""
    expr_list = list(expr)
    for i in range(len(expr_list)):
        if random.random() < MUTATION_RATE:
            expr_list[i] = random.choice(['x', '+', '-', '*', '/'])
    return ''.join(expr_list)

def crossover(parent1, parent2):
    """Single-point crossover."""
    if len(parent1) < 2 or len(parent2) < 2:
        return parent1, parent2
    cut1 = random.randint(1, len(parent1) - 1)
    cut2 = random.randint(1, len(parent2) - 1)
    child1 = parent1[:cut1] + parent2[cut2:]
    child2 = parent2[:cut2] + parent1[cut1:]
    return child1, child2

def tournament_selection(pop, fits):
    """Select the best of k random individuals."""
    chosen = random.sample(list(zip(pop, fits)), TOURNAMENT_SIZE)
    return min(chosen, key=lambda x: x[1])[0]

# ---------------------
# Main GEP loop
# ---------------------

def gene_expression_optimization():
    population = [random_gene() for _ in range(POP_SIZE)]

    for gen in range(GENERATIONS):
        fitnesses = [fitness(expr) for expr in population]
        best_idx = min(range(len(fitnesses)), key=lambda i: fitnesses[i])
        best_expr = population[best_idx]
        best_fit = fitnesses[best_idx]

        if gen % 10 == 0 or gen == GENERATIONS - 1:
            print(f"Generation {gen:3d} | Best Expr: {best_expr} | Fitness = {best_fit:.6f}")

        new_pop = [best_expr]  # elitism
        while len(new_pop) < POP_SIZE:
            p1 = tournament_selection(population, fitnesses)
            p2 = tournament_selection(population, fitnesses)
            c1, c2 = crossover(p1, p2)
            new_pop.extend([mutate(c1), mutate(c2)])
        population = new_pop[:POP_SIZE]
```

```python
def gene_expression_optimization():
    population = [random_gene() for _ in range(POP_SIZE)]

    for gen in range(GENERATIONS):
        fitnesses = [fitness(expr) for expr in population]
        best_idx = min(range(len(fitnesses)), key=lambda i: fitnesses[i])
        best_expr = population[best_idx]
        best_fit = fitnesses[best_idx]

        if gen % 10 == 0 or gen == GENERATIONS - 1:
            print(f"Generation {gen:3d} | Best Expr: {best_expr} | Fitness = {best_fit:.6f}")

        new_pop = [best_expr]  # elitism
        while len(new_pop) < POP_SIZE:
            p1 = tournament_selection(population, fitnesses)
            p2 = tournament_selection(population, fitnesses)
            c1, c2 = crossover(p1, p2)
            new_pop.extend([mutate(c1), mutate(c2)])
        population = new_pop[:POP_SIZE]

    print("\n✅ Optimization Complete!")
    print(f"Best evolved expression: {best_expr}")
    return best_expr

# ---------------------
# Run
# ---------------------
if __name__ == "__main__":
    best = gene_expression_optimization()

    print("\nSample comparison:")
    for x in [-3, 0, 2, 5]:
        print(f"x={x:>2} | Target={target_function(x):>8.3f} | Evolved={evaluate_expression(best, x):>8.3f}")
```

```
Generation   0 | Best Expr: x | Fitness = 2487.000000
Generation  10 | Best Expr: x | Fitness = 2487.000000
Generation  20 | Best Expr: x | Fitness = 2487.000000
Generation  30 | Best Expr: x | Fitness = 2487.000000
Generation  40 | Best Expr: x | Fitness = 2487.000000
Generation  50 | Best Expr: x | Fitness = 2487.000000
Generation  60 | Best Expr: x | Fitness = 2487.000000
Generation  70 | Best Expr: x | Fitness = 2487.000000
Generation  80 | Best Expr: x | Fitness = 2487.000000
Generation  90 | Best Expr: x | Fitness = 2487.000000
Generation  99 | Best Expr: x | Fitness = 2487.000000

✅ Optimization Complete!
Best evolved expression: x

Sample comparison:
x=-3 | Target=   7.000 | Evolved=  -3.000
x= 0 | Target=   1.000 | Evolved=   0.000
x= 2 | Target=   7.000 | Evolved=   2.000
x= 5 | Target=  31.000 | Evolved=   5.000
```