**Optimization: Dead Code Elimination**

**Original Code:**

In the TraverseTree method:

if (root.token.lex == "elseif" || root.token.lex == "else" && !lastCondition && begin) {

stop = false;

lastCondition = true;

}

If stop or lastCondition is reset multiple times unnecessarily in this block, it can be simplified to avoid redundant operations.

**Optimized Code:**

if (root.token.lex == "elseif" || root.token.lex == "else" && !lastCondition && begin) {

if (!stop || !lastCondition) {

stop = false;

lastCondition = true;

}

}  
  
**Optimization: Strength Reduction**

**Original Code:**

In GetOpWeight:

int w = 0;

switch (op) {

case "+":

case "-":

w = 1;

break;

case "\*":

case "/":

w = 2;

break;

}

Using a Dictionary lookup can reduce the overhead of a switch for this small set of operations.

**Optimized Code:**

private static readonly Dictionary<string, int> operatorWeights = new Dictionary<string, int> {

{ "+", 1 },

{ "-", 1 },

{ "\*", 2 },

{ "/", 2 }

};

public static int GetOpWeight(string op) {

return operatorWeights.TryGetValue(op, out int weight) ? weight : 0;

}  
  
**Optimization: Common Subexpression Elimination**

**Original Code:**

**In EvaluateExp:**

for (int i = 0; i < postfixExp.Count; i++) {

if (postfixExp[i] == "+" || postfixExp[i] == "-" || postfixExp[i] == "\*" || postfixExp[i] == "/") {

int op2 = stk.Pop();

int op1 = stk.Pop();

int res = ApplyOperator(op1, op2, postfixExp[i]);

stk.Push(res);

}

}

The expression postfixExp[i] is accessed multiple times within the loop.

**Optimized Code:**

for (int i = 0; i < postfixExp.Count; i++) {

string token = postfixExp[i];

if (token == "+" || token == "-" || token == "\*" || token == "/") {

int op2 = stk.Pop();

int op1 = stk.Pop();

int res = ApplyOperator(op1, op2, token);

stk.Push(res);

}

}

**Optimization: Loop-Invariant Code Motion**

**Original Code:**

**In InfixToPostfix:**

while (stk.Count > 0 && stk.Peek() != "(" && HasHigherPre(stk.Peek(), infix[i])) {

result.Add(stk.Peek());

stk.Pop();

}

The condition stk.Peek() is evaluated repeatedly in the loop but does not change during each iteration.

**Optimized Code:**

string top;

while (stk.Count > 0 && (top = stk.Peek()) != "(" && HasHigherPre(top, infix[i])) {

result.Add(top);

stk.Pop();

}