

Topic: Implement constant folding optimization on input 3-address code block  in form of quadruples. ( In C / C++ / Java).

|  |  |
| --- | --- |
| Name | Imaad Hajwane |
| SRN No | 202101132 |
| Roll no | 23 |
| Division / Batch | A-1 |
| Assignment no | 6 |
| Subject | CD Lab |

# Java Code:

import java.util.\*;

// Class to represent quadruple structure

class Quadruple {

    String op;

    String arg1;

    String arg2;

    String result;

    Quadruple(String op, String arg1, String arg2, String result) {

        this.op = op;

        this.arg1 = arg1;

        this.arg2 = arg2;

        this.result = result;

    }

}

public class Main {

    // Function to perform constant folding optimization

    public static List<Quadruple> constantFolding(List<Quadruple> quadruples) {

        List<Quadruple> optimizedQuadruples = new ArrayList<>();

        Stack<Map.Entry<String, Integer>> constantStack = new Stack<>();

        for (Quadruple quad : quadruples) {

            int value1 = 0, value2 = 0;

            boolean isConst1 = false, isConst2 = false;

            // Check if argument 1 is a constant

            if (Character.isDigit(quad.arg1.charAt(0))) {

                isConst1 = true;

                value1 = Integer.parseInt(quad.arg1);

            }

            // Check if argument 2 is a constant

            if (Character.isDigit(quad.arg2.charAt(0))) {

                isConst2 = true;

                value2 = Integer.parseInt(quad.arg2);

            }

            // Evaluate constant expression if possible

            if (isConst1 && isConst2) {

                int result;

                switch (quad.op) {

                    case "+":

                        result = value1 + value2;

                        break;

                    case "-":

                        result = value1 - value2;

                        break;

                    case "\*":

                        result = value1 \* value2;

                        break;

                    case "/":

                        if (value2 != 0)

                            result = value1 / value2;

                        else {

                            System.err.println("Division by zero!");

                            System.exit(1);

                            result = 0; // unreachable

                        }

                        break;

                    default:

                        optimizedQuadruples.add(quad); // If operation is not supported for constant folding

                        continue;

                }

                // Push the result onto the constant stack

                constantStack.push(new AbstractMap.SimpleEntry<>(quad.result, result));

            } else {

                // If one or both arguments are not constants, push the quadruple onto the stack

                optimizedQuadruples.add(quad);

            }

        }

        // Pop constants from the stack and replace references with their values

        while (!constantStack.empty()) {

            Map.Entry<String, Integer> constant = constantStack.pop();

            for (Quadruple quad : optimizedQuadruples) {

                if (quad.arg1.equals(constant.getKey()))

                    quad.arg1 = String.valueOf(constant.getValue());

                if (quad.arg2.equals(constant.getKey()))

                    quad.arg2 = String.valueOf(constant.getValue());

            }

        }

        return optimizedQuadruples;

    }

    // Function to print quadruples

    public static void printQuadruples(List<Quadruple> quadruples) {

        for (Quadruple quad : quadruples) {

            System.out.println("(" + quad.op + ", " + quad.arg1 + ", " + quad.arg2 + ", " + quad.result + ")");

        }

    }

    public static void main(String[] args) {

        // Example input 3-address code in form of quadruples

        List<Quadruple> quadruples = new ArrayList<>(Arrays.asList(

                new Quadruple("+", "2", "8", "t1"),

                new Quadruple("\*", "2", "t1", "t2"),

                new Quadruple("+", "t2", "4", "t3"),

                new Quadruple("/", "t3", "7", "t4"),

                new Quadruple("+", "t4", "t4", "t5")

        ));

        System.out.println("Original Quadruples:");

        printQuadruples(quadruples);

        // Perform constant folding optimization

        List<Quadruple> optimizedQuadruples = constantFolding(quadruples);

        System.out.println("\nOptimized Quadruples:");

        printQuadruples(optimizedQuadruples);

    }

}

**Output:-**

****