Date : 26/04/2022

CSPC62 : COMPILER DESIGN
LAB-8

Roll no.: **106119100**

Name: **Rajneesh Pandey**

Section: **CSE-B**

Perform local optimization on a basic block.

Code:

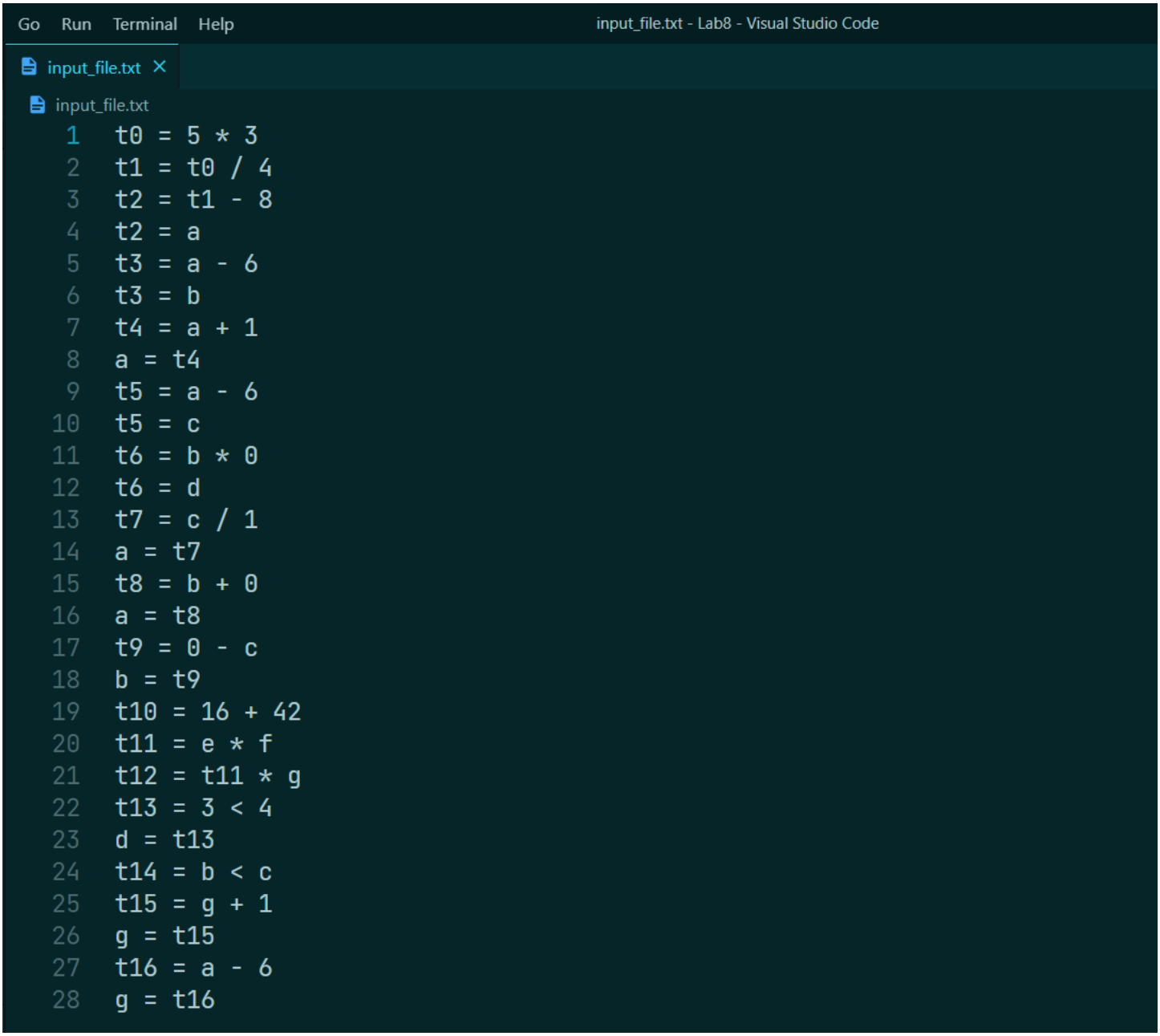
Optimize_ICG.cpp

```
import re

def isTemporary(s):
    return bool(re.match(r"^t[0-9]*$", s))


def isIdentifier(s):
    return bool(re.match(r"^[A-Za-z][A-Za-z0-9_]*$", s))


def showICG(allLines):
    for line in allLines:
        print("\t",line.strip())


def createSubexpressions(allLines):
    expressions = {}
    variables = {}
    for line in allLines:
        tokens = line.split()
        if len(tokens) == 5:
            if tokens[0] in variables and variables[tokens[0]] in expressions:
                print(tokens[0], variables[tokens[0]], expressions[variables[tokens[0]]])
                del expressions[variables[tokens[0]]]
            expressionRHS = tokens[2] + " " + tokens[3] + " " + tokens[4]
            if expressionRHS not in expressions:
                expressions[expressionRHS] = tokens[0]
                if isIdentifier(tokens[2]):
                    variables[tokens[2]] = expressionRHS
                if isIdentifier(tokens[4]):
                    variables[tokens[4]] = expressionRHS
    return expressions


def eliminateCommonSubexpressions(allLines):
    expressions = createSubexpressions(allLines)
    updatedAllLines = allLines[:]
    for i in range(len(allLines)) :
        tokens = allLines[i].split()
        if len(tokens) == 5 :
            expressionRHS = tokens[2] + " " + tokens[3] + " " + tokens[4]
            if expressionRHS in expressions and expressions[expressionRHS] != tokens[0]:
                updatedAllLines[i] = tokens[0] + " " + tokens[1] + " " +
expressions[expressionRHS]
    return updatedAllLines
```

```
def evaluateExpression(expression) :
    tokens = expression.split()
    if len(tokens) != 5 :
        return expression
    acceptedOperators = {"+", "-", "*", "/", "*", "&", "|", "^", "==", ">=", "<=", "!=", ">",
"<"}
    if tokens[1] != "=" or tokens[3] not in acceptedOperators:
        return expression
    if tokens[2].isdigit() and tokens[4].isdigit() :
        return " ".join([tokens[0], tokens[1], str(eval(str(tokens[2] + tokens[3] +
tokens[4])))])
    if tokens[2].isdigit() or tokens[4].isdigit() : #Replace the identifier with a number and
evaluate
        op1 = "5" if isIdentifier(tokens[2]) else tokens[2]
        op2 = "5" if isIdentifier(tokens[4]) else tokens[4]
        op = tokens[3]
        try :
            result = int(eval(op1+op+op2))
            if result == 0 : #multiplication with 0
                return " ".join([tokens[0],tokens[1], "0"])
            elif result == 5 : # add zero, subtract 0, multiply 1, divide 1
                if isIdentifier(tokens[2]) and tokens[4].isdigit() :
                    return " ".join([tokens[0], tokens[1], tokens[2]])
                elif isIdentifier(tokens[4]) and tokens[2].isdigit():
                    return " ".join([tokens[0], tokens[1], tokens[4]])
            elif result == -5 and tokens[2] == "0" : # 0 - id
                return " ".join([tokens[0], tokens[1], "-"+tokens[4]])
            return " ".join(tokens)
        except NameError :
            return expression
        except ZeroDivisionError :
            print("Division By Zero!")
            quit()
    return expression


def constantFolding(allLines) :
    updatedAllLines = []
    for line in allLines :
        updatedAllLines.append(evaluateExpression(line))
    return updatedAllLines


def deadCodeElimination(allLines) :
    num_lines = len(allLines)
    definedTempVars = set()
    for line in allLines :
        tokens = line.split()
        if isTemporary(tokens[0]) :
            definedTempVars.add(tokens[0])
    usefulTempVars = set()
    for line in allLines :
        tokens = line.split()
        if len(tokens) >= 2 :
```

```
            if isTemporary(tokens[1]) :
                usefulTempVars.add(tokens[1])
        if len(tokens) >= 3 :
            if isTemporary(tokens[2]) :
                usefulTempVars.add(tokens[2])
    unwantedTempVars = definedTempVars - usefulTempVars
    updatedAllLines = []
    for line in allLines :
        tokens = line.split()
        if tokens[0] not in unwantedTempVars :
            updatedAllLines.append(line)
    if num_lines == len(updatedAllLines) :
        return updatedAllLines
    return deadCodeElimination(updatedAllLines)


if __name__ == "__main__":
    allLines = []
    f = open("input_file.txt", "r")
    for line in f:
        allLines.append(line)
    f.close()

    print("\n")

    # Input
    print("Generated ICG given as input for optimization: \n")
    showICG(allLines)
    print("\n")

    # Elimination of Common Subexpressions
    icgAfterEliminationOfCommonSubexpressions = eliminateCommonSubexpressions(allLines)
    print("ICG after eliminating common subexpressions: \n")
    showICG(icgAfterEliminationOfCommonSubexpressions)
    print("\n")

    # Constant folding
    icgAfterConstantFolding = constantFolding(icgAfterEliminationOfCommonSubexpressions)
    print("ICG after constant folding: \n")
    showICG(icgAfterConstantFolding)
    print("\n")

    # Dead Code Elimination
    icgAfterDeadCodeElimination = deadCodeElimination(icgAfterConstantFolding)
    print("Optimized ICG after dead code elimination: \n")
    showICG(icgAfterDeadCodeElimination)
    print("\n")

    print("Optimization done by eliminating", len(allLines)-len(icgAfterDeadCodeElimination),
"lines.")
    print("\n")
```

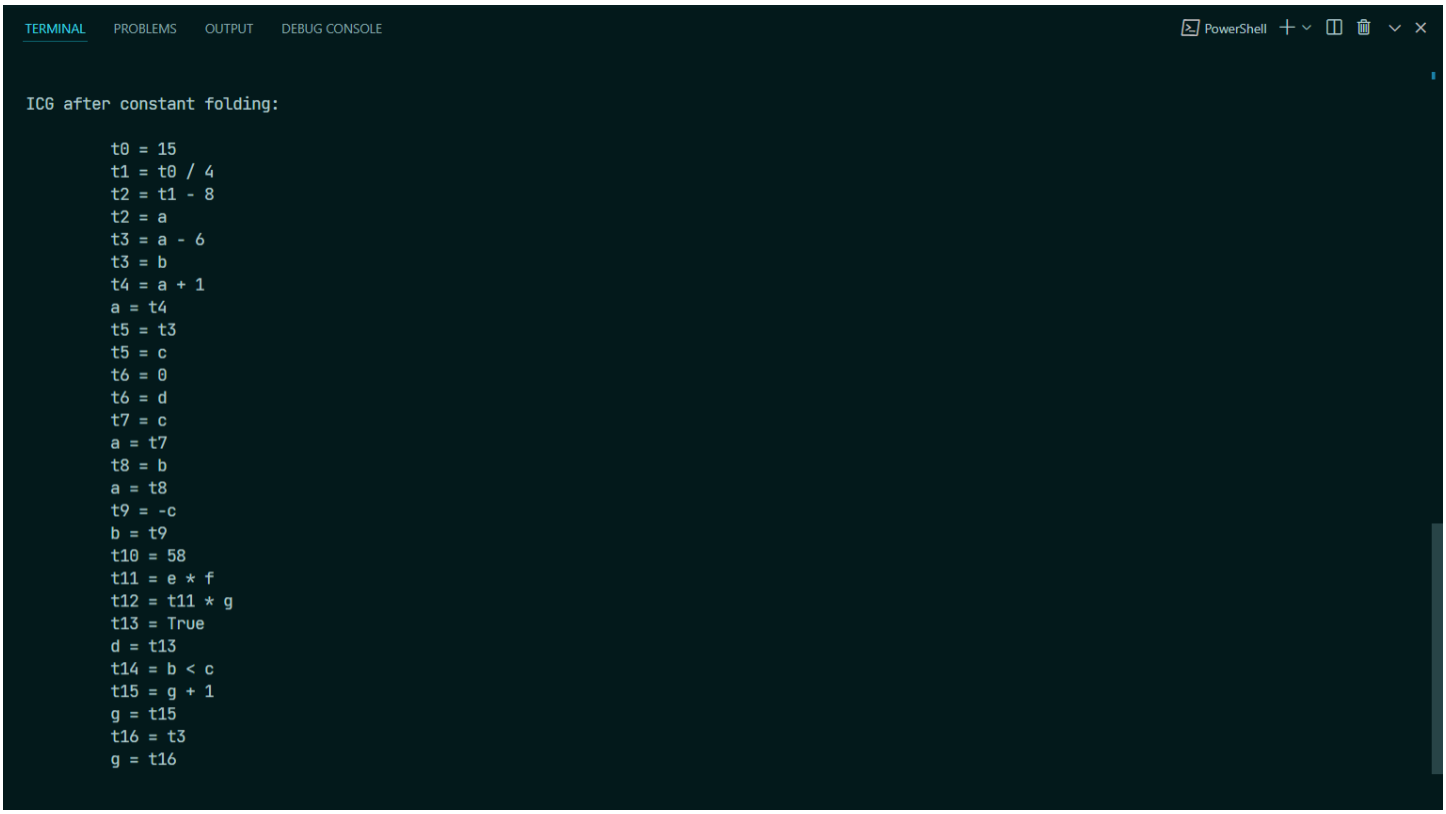
input_file.txt

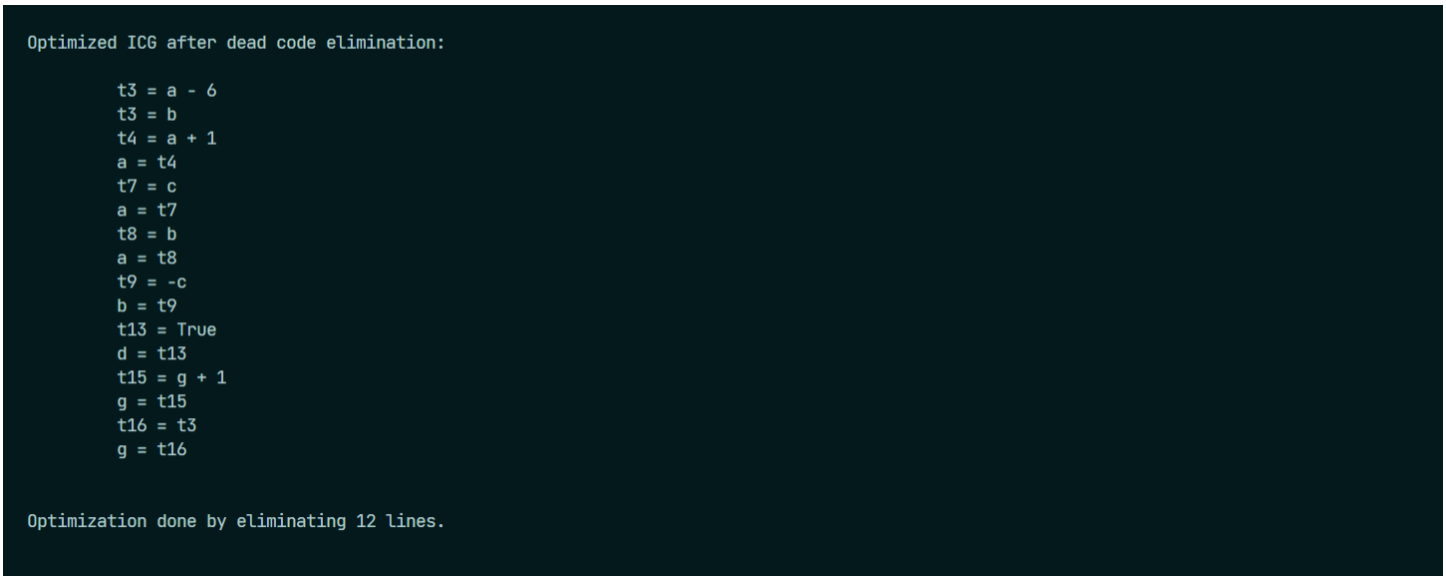


Output:

```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE                                PowerShell

Shell integration activated
PS D:\Documents\Academics\NIT Trichy\Semesters\VI-Semester\Compiler Design\Lab\Lab8> python '.\Optimize ICG.py'


Generated ICG given as input for optimization:

        t0 = 5 * 3
        t1 = t0 / 4
        t2 = t1 - 8
        t2 = a
        t3 = a - 6
        t3 = b
        t4 = a + 1
        a = t4
        t5 = a - 6
        t5 = c
        t6 = b * 0
        t6 = d
        t7 = c / 1
        a = t7
        t8 = b + 0
        a = t8
        t9 = 0 - c
        b = t9
        t10 = 16 + 42
        t11 = e * f
        t12 = t11 * g
        t13 = 3 < 4
        d = t13
        t14 = b < c
        t15 = g + 1
        g = t15
        t16 = a - 6
        g = t16
```

```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE                                PowerShell

ICG after eliminating common subexpressions:

        t0 = 5 * 3
        t1 = t0 / 4
        t2 = t1 - 8
        t2 = a
        t3 = a - 6
        t3 = b
        t4 = a + 1
        a = t4
        t5 = t3
        t5 = c
        t6 = b * 0
        t6 = d
        t7 = c / 1
        a = t7
        t8 = b + 0
        a = t8
        t9 = 0 - c
        b = t9
        t10 = 16 + 42
        t11 = e * f
        t12 = t11 * g
        t13 = 3 < 4
        d = t13
        t14 = b < c
        t15 = g + 1
        g = t15
        t16 = t3
        g = t16
```


TERMINAL
PROBLEMS
OUTPUT
DEBUG CONSOLE
PowerShell
ICG after constant folding:
t0 = 15
t1 = t0 / 4
t2 = t1 - 8
t2 = a
t3 = a - 6
t3 = b
t4 = a + 1
a = t4
t5 = t3
t5 = c
t6 = 0
t6 = d
t7 = c
a = t7
t8 = b
a = t8
t9 = -c
b = t9
t10 = 58
t11 = e * f
t12 = t11 * g
t13 = True
d = t13
t14 = b < c
t15 = g + 1
g = t15
t16 = t3
g = t16


Optimized ICG after dead code elimination:
t3 = a - 6
t3 = b
t4 = a + 1
a = t4
t7 = c
a = t7
t8 = b
a = t8
t9 = -c
b = t9
t13 = True
d = t13
t15 = g + 1
g = t15
t16 = t3
g = t16
Optimization done by eliminating 12 lines.