

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Date : 26/04/2022 | | | | | | | |
|  | CSPC62 : COMPILER DESIGN  **LAB-8** | | | | | |  |
|  |  | | | | |  | |
|  | | |  |  | | | |
|  | | | Roll no.: 106119100Name: Rajneesh PandeySection: 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")

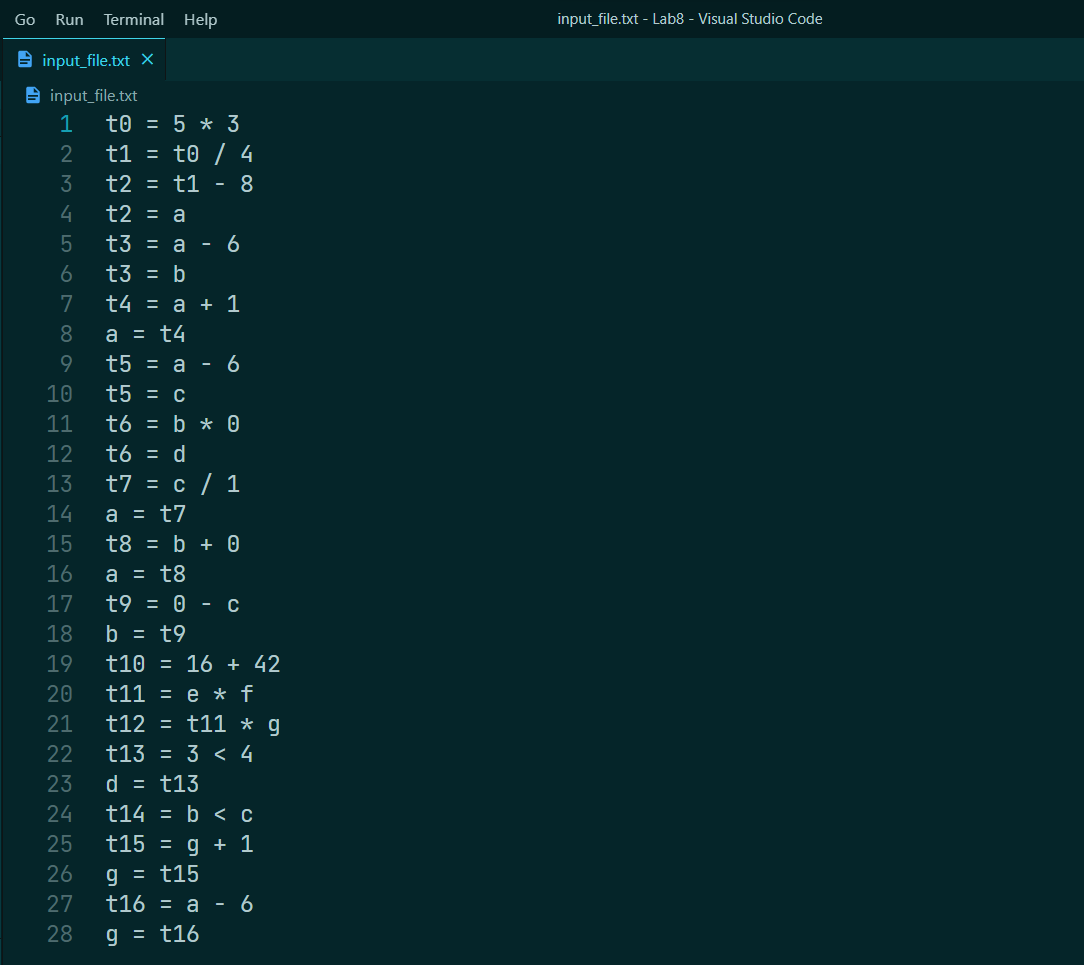
    showICG(icgAfterDeadCodeElimination)

    print("\n")

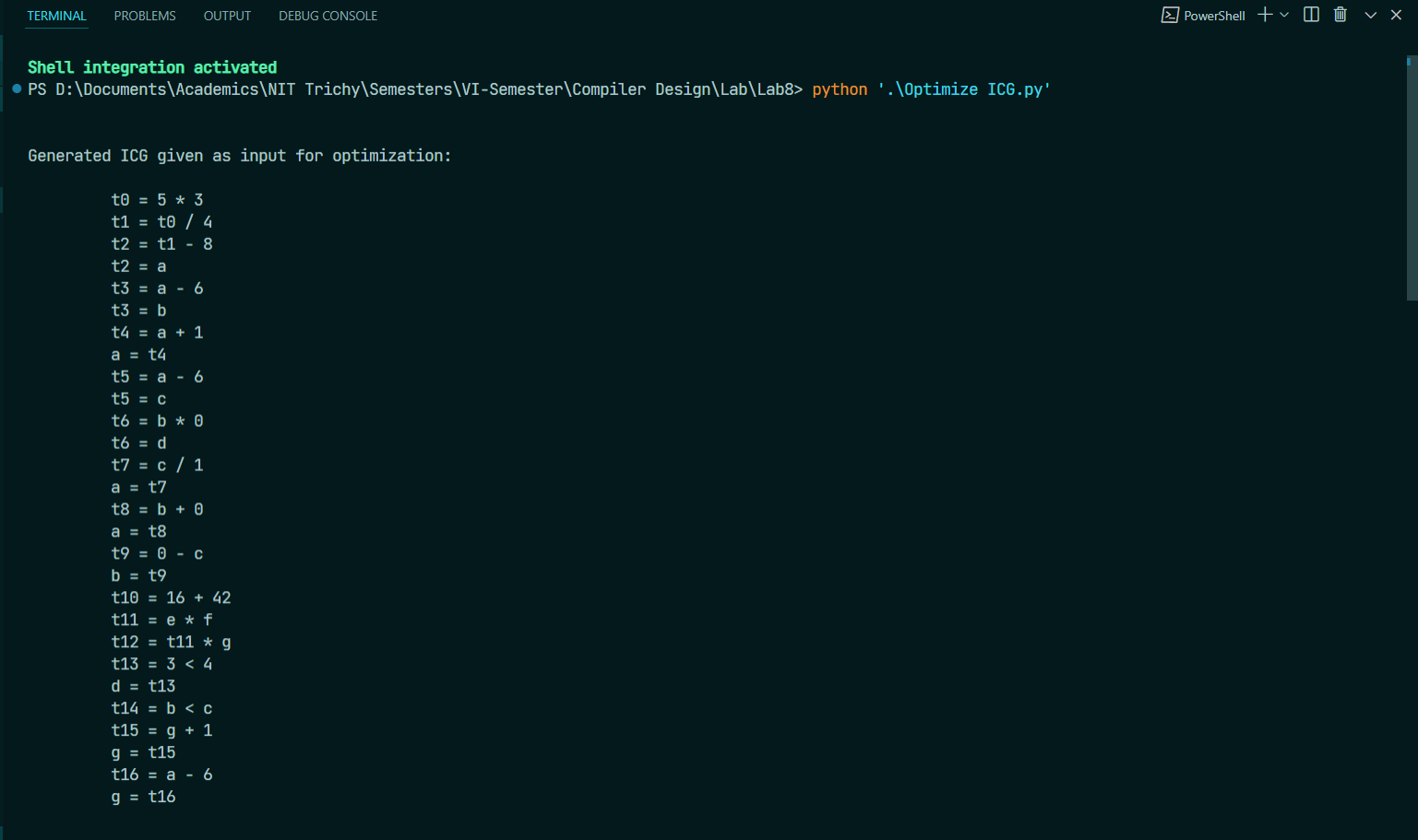
    print("Optimization done by eliminating", len(allLines)**-**len(icgAfterDeadCodeElimination), "lines.")

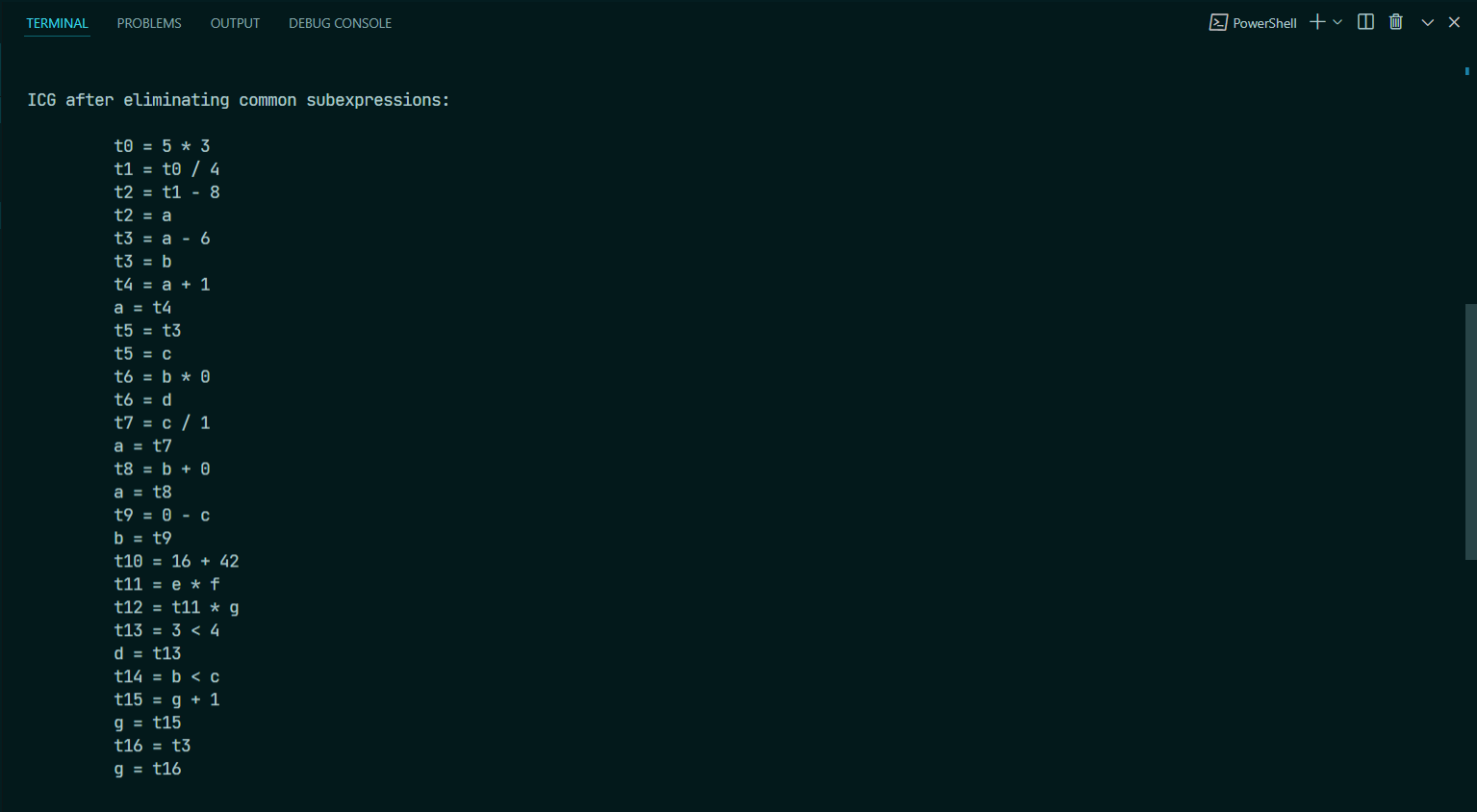
    print("\n")

input\_file.txt



Output:







A picture containing shape

Description automatically generated