Shri Ramdeobaba College of Engineering and Management, Nagpur
Department of Computer Science and Engineering
Session: 2021-2022 [EVEN SEM]

Compiler Design Lab

Name : Janak Mandavgade

Sec : A

Roll no. : 43

Batch : A3

Subject : Compiler Design

PRACTICAL No. 7

Aim:

Write a code to implement Local optimization techniques until no further optimization is possible for the given three address code.

Code :

```
def optimize_TAC(tac):
    def copy_propagation(tac):
        var_map = {}
        optimized_tac = []

        for line in tac:
            tokens = line.split()

            if len(tokens) == 3 and tokens[1] == "=":
                var_map[tokens[0]] = tokens[2]
            else:
                new_line = " ".join([var_map.get(token, token) for token in tokens])
                optimized_tac.append(new_line)
```

```
    return optimized_tac

def constant_propagation(tac):
    constants = {}
    optimized_tac = []

    for line in tac:
        tokens = line.split()

        if len(tokens) == 3 and tokens[1] == "=" and tokens[2].isdigit():
            constants[tokens[0]] = tokens[2]
        else:
            new_line = " ".join([constants.get(token, token) for token in tokens])
            optimized_tac.append(new_line)

    return optimized_tac

def constant_folding(tac):
    optimized_tac = []

    for line in tac:
        tokens = line.split()

        if len(tokens) == 5:
```

```
            op1, operator, op2 = tokens[2], tokens[3], tokens[4]

            if op1.isdigit() and op2.isdigit():
                result = eval(f"{op1} {operator} {op2}")
                new_line = f"{tokens[0]} = {result}"
            else:
                new_line = line

            optimized_tac.append(new_line)
        else:
            optimized_tac.append(line)

    return optimized_tac

def common_subexpression_elimination(tac):
    subexpr_map = {}
    optimized_tac = []

    for line in tac:
        tokens = line.split()

        if len(tokens) == 5:
            subexpr = " ".join(tokens[2:])

            if subexpr in subexpr_map:
```

```
                new_line = f"{tokens[0]} = {subexpr_map[subexpr]}"
            else:
                subexpr_map[subexpr] = tokens[0]
                new_line = line

            optimized_tac.append(new_line)
        else:
            optimized_tac.append(line)

    return optimized_tac

optimized_tac = tac
prev_tac = []

while prev_tac != optimized_tac:
    prev_tac = optimized_tac
    optimized_tac = copy_propagation(optimized_tac)
    optimized_tac = constant_propagation(optimized_tac)
    optimized_tac = constant_folding(optimized_tac)
    optimized_tac = common_subexpression_elimination(optimized_tac)

return optimized_tac


tac = [
```

```
    # "a = 5",
    # "b = a",
    # "c = 3",
    # "d = b + c",
    # "e = 5 + 3",
    # "f = d * e",
    # "g = b + c",
    # "h = f + g"

    "a = 2",
    "b = x * x",
    "c = x",
    "b = a + 5 ",
    "e = b + c",
    "f = c * c",
    "g = d + e",
    "h = e * f"
]

optimized_tac = optimize_TAC(tac)

print("Original TAC:")
print("\n".join(tac))

print("\nOptimized TAC:")
```

```
print("\n".join(optimized_tac))
```

Input:

```
"a = 2",
"b = x * x",
"c = x",
"b = a + 5 ",
"e = b + c",
"f = c * c",
"g = d + e",
"h = e * f"
```

Output: