Shri Ramdeobaba College of Engineering and Management, Nagpur
Department of Computer Science and Engineering
Session: 2024-2025

Compiler Design Lab

PRACTICAL No. 6

Name: Shreyas Sahare

Roll No : 60

Batch : A4

Topic: Code Optimization

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

Input: Three Address Code (non-optimized)

Implementation: Identify and apply local optimization techniques to optimize the TAC

- Copy Propagation
- Constant propagation
- Constant Folding
- Common Subexpression Elimination
- Dead code elimination

Output: Optimized Three Address Code

Input Code:

```
def copy_propagation(tac):
  # Replace variables with their copies
  for i in range(len(tac)):
      if tac[i].startswith('t') and tac[i].count('=') == 1:
          var, expr = tac[i].split('=')
          var = var.strip()
          expr = expr.strip()
          if expr.startswith('t') and expr.count('=') == 0:
              tac[i] = f"{var} = {expr}"
  return tac


def constant_propagation(tac):
  # Replace variables with their constant values
  constants = {}
  for i in range(len(tac)):
      if tac[i].startswith('t') and tac[i].count('=') == 1:
          var, expr = tac[i].split('=')
          var = var.strip()
          expr = expr.strip()
          if expr.isdigit():
              constants[var] = int(expr)
          elif expr in constants:
              tac[i] = f"{var} = {constants[expr]}"
  return tac


def constant_folding(tac):
  # Simplify expressions involving constants
```

```
  for i in range(len(tac)):
      if tac[i].startswith('t') and tac[i].count('=') == 1:
          var, expr = tac[i].split('=')
          var = var.strip()
          expr = expr.strip()
          if '+' in expr:
              left, right = expr.split('+')
              left = left.strip()
              right = right.strip()
              if left.isdigit() and right.isdigit():
                  tac[i] = f"{var} = {int(left) + int(right)}"
  return tac


def common_subexpression_elimination(tac):
  # Eliminate redundant computations
  expressions = {}
  for i in range(len(tac)):
      if tac[i].startswith('t') and tac[i].count('=') == 1:
          var, expr = tac[i].split('=')
          var = var.strip()
          expr = expr.strip()
          if expr in expressions:
              tac[i] = f"{var} = {expressions[expr]}"
          else:
              expressions[expr] = var
  return tac


def dead_code_elimination(tac):
  # Remove code that does not affect the program's output
```

```
  used_vars = set()
  for i in range(len(tac)):
      if tac[i].startswith('t') and tac[i].count('=') == 1:
          var, expr = tac[i].split('=')
          var = var.strip()
          expr = expr.strip()
          if expr.startswith('t'):
              used_vars.add(expr)
  return [line for line in tac if line.split('=')[0].strip() in used_vars]


def optimize_tac(tac):
  tac = copy_propagation(tac)
  tac = constant_propagation(tac)
  tac = constant_folding(tac)
  tac = common_subexpression_elimination(tac)
  tac = dead_code_elimination(tac)
  return tac


# Example non-optimized TAC
tac = [
  "t1 = 5",
  "t2 = 10",
  "t3 = t1 + t2",
  "t4 = t1 + t2",
  "t5 = t3 + t4",
  "t6 = t5 + 1",
  "t7 = t6",
  "t8 = t7 + 2",
  "t9 = t8"
```

```
]

# Optimize the TAC
optimized_tac = optimize_tac(tac)

# Print the optimized TAC
for line in optimized_tac:
  print(line)
```

Output:

```
t3 = t1 + t2
t6 = t5 + 1
t8 = t7 + 2

=== Code Execution Successful ===
```