Name – Saurabh Khandagale

Roll No – 46

PRACTICAL NO 7

AIM - Write a program to find the data flow equations by considering a program flow graph.

CODE –

```
from copy import deepcopy


def get_blocks():
    blocks = []
    temp = []

    lastFlag = 0

    # rule 1
    temp.append(TAC[1])

    for i in range(2, len(TAC)):
        # rule 2
        if TAC[i].startswith('if'):
            lastFlag = 1
            temp1 = deepcopy(temp)
            blocks.append(temp1)
            temp.clear()
            temp.append(TAC[i])
        elif 'goto' in TAC[i]:
            lastFlag = 1
            temp.append(TAC[i])
        # rule 3
        elif lastFlag == 1:
            temp1 = deepcopy(temp)
            blocks.append(temp1)
            temp.clear()
            temp.append(TAC[i])
            lastFlag = 0
        else:
            lastFlag = 0
            temp.append(TAC[i])

    blocks.append(temp)

    # print the blocks
    print("\n"+"*"*10+"Blocks"+"*"*10+"\n")
    for i in range(len(blocks)):
        print("Block {}:".format(i+1))
        for j in range(len(blocks[i])):
            print("\t"+blocks[i][j])
        print('\n')

    return blocks

def control_flow(blocks):
    flow = {}
    # initializing the dict for string control flow
    for i in range(len(blocks)):
        flow["B"+str(i+1)] = ['B'+str(i+1)]

    gotoFlag = 9999
    for i in range(len(blocks)):
        if blocks[i][-1].startswith('goto'):
            gotoFlag = i+1
            for j in range(1, i+1):
                flow["B"+str(i+1)].insert(0, 'B'+str(j))
        else:
            for j in range(1, i+1):
                if j != gotoFlag:
                    flow["B"+str(i+1)].insert(0, 'B'+str(j))

    for i in flow:
        flow[i].sort()

    print("*"*10+"Program flow graph"+"*"*10+"\n")
    for i in flow:
        print(i, '->', flow[i])

    return flow

def gen_kill(blocks, TAC):
    counter = 1

    # initializing empty dictionaries for gen and kill
    gen = {}
    kill = {}
    for i in range(len(blocks)):
        gen["B"+str(i+1)] = []
        kill["B"+str(i+1)] = []
```

```
for i in range(len(blocks)):
    for j in blocks[i]:
        if "=" in j:
            value = j.split("=")
            gen["B"+str(i+1)].append(counter)
            for k in range(len(TAC)):
                if TAC[k] != j:
                    if value[0] in TAC[k]:
                        if "=" in TAC[k]:
                            value1 = TAC[k].split("=")
                            if value1[0] == value[0]:
                                kill["B"+str(i+1)].append(k)
        counter += 1


# printing gen
print("\n"+"*"*10+"Gen"+"*"*10+"\n")
for i in gen:
    if len(gen[i]) == 0:
        print(i, '->', ["∅"])
    else:
        print(i, '->', gen[i])


# printing kill
print("\n"+"-"*10+"Kill"+"-"*10+"\n")
for i in kill:
    if len(kill[i]) == 0:
        print(i, '->', ["∅"])
    else:
        print(i, '->', kill[i])
    return gen, kill


def get_predecessors():
    return


def in_out(blocks, gen, kill):
    predecessors = {
        "B1": [],
        "B2": ["B1", "B3"],
        "B3": ["B2"],
        "B4": ["B2"],
    }
    IN = {}
    OUT = {}
    prev_IN = {}
    prev_OUT = {}
    iterations = 0

    # converting gen and kill to sets
    for i in gen:
        gen[i] = set(gen[i])
    for i in kill:
        kill[i] = set(kill[i])

    # initilizing in and out as empty sets
    for i in range(len(blocks)):
        IN["B"+str(i+1)] = set()
        OUT["B"+str(i+1)] = gen["B"+str(i+1)]
        prev_IN["B"+str(i+1)] = set()
        prev_OUT["B"+str(i+1)] = set()
```

```
while prev_IN != IN or prev_OUT != OUT:
    print("\n"+"*"*10+"Iteration {}".format(iterations)+"*"*10)
    # print IN
    print("\n"+"-"*10+"IN"+"-"*10+"\n")
    for i in IN:
        if len(IN[i]) == 0:
            print(i, '->', ["Ø"])
        else:
            print(i, '->', IN[i])

    # print OUT
    print("\n"+"*"*10+"OUT"+"*"*10+"\n")
    for i in OUT:
        if len(OUT[i]) == 0:
            print(i, '->', ["Ø"])
        else:
            print(i, '->', OUT[i])

    prev_OUT = deepcopy(OUT)
    prev_IN = deepcopy(IN)

    for i in range(len(blocks)):
        OUT["B"+str(i+1)] = prev_OUT["B"+str(i+1)].union((IN["B"+str(i+1)] -
                                                kill["B"+str(i+1)]).union(gen["B"+str(i+1)]))
        temp = set()
        for x in predecessors["B"+str(i+1)]:
            temp = temp.union(OUT[x])
        IN["B"+str(i+1)] = prev_IN["B"+str(i+1)].union(temp)
    iterations += 1
    return IN, OUT


with open('input.txt', 'r') as f:
    f = open('input.txt', 'r')
    TAC = f.read().split('\n')
    TAC.insert(0, 'start')

    # blocks
    blocks = get_blocks()

    # control flow
    flow = control_flow(blocks=blocks)

    # gen and kill
    gen, kill = gen_kill(blocks=blocks, TAC=TAC)

    # in and out
    IN, OUT = in_out(blocks=blocks, gen=gen, kill=kill)
```

OUTPUT –

```
**********Blocks**********

Block 1:
            sum = 0
            i = 0


Block 2:
            if i>n goto 12


Block 3:
            T1 = addr(a)
            T2 = i*4
            T3 = T1[T2]
            T4 = sum + T3
            sum = T4
            T5 = i + 1
            i = T5
            goto 3


Block 4:
            return sum


**********Program flow graph**********

B1 -> ['B1']
B2 -> ['B1', 'B2']
B3 -> ['B1', 'B2', 'B3']
B4 -> ['B1', 'B2', 'B4']

**********Gen**********

B1 -> [1, 2]
B2 -> ['Ø']
B3 -> [4, 5, 6, 7, 8, 9, 10]
B4 -> ['Ø']

----------Kill----------

B1 -> [8, 10]
B2 -> ['Ø']
B3 -> [1, 2]
B4 -> ['Ø']

**********Iteration 0**********

----------IN----------

B1 -> ['Ø']
B2 -> ['Ø']
B3 -> ['Ø']
B4 -> ['Ø']

**********OUT**********

B1 -> {1, 2}
B2 -> ['Ø']
B3 -> {4, 5, 6, 7, 8, 9, 10}
```

```
B4 -> ['Ø']

**********Iteration 1**********

----------IN----------

B1 -> ['Ø']
B2 -> {1, 2, 4, 5, 6, 7, 8, 9, 10}
B3 -> ['Ø']
B4 -> ['Ø']

**********OUT**********

B1 -> {1, 2}
B2 -> ['Ø']
B3 -> {4, 5, 6, 7, 8, 9, 10}
B4 -> ['Ø']

**********Iteration 2**********

----------IN----------

B1 -> ['Ø']
B2 -> {1, 2, 4, 5, 6, 7, 8, 9, 10}
B3 -> {1, 2, 4, 5, 6, 7, 8, 9, 10}
B4 -> {1, 2, 4, 5, 6, 7, 8, 9, 10}

**********OUT**********

B1 -> {1, 2}
B2 -> {1, 2, 4, 5, 6, 7, 8, 9, 10}
B3 -> {4, 5, 6, 7, 8, 9, 10}
B4 -> ['Ø']

**********Iteration 3**********

----------IN----------

B1 -> ['Ø']
B2 -> {1, 2, 4, 5, 6, 7, 8, 9, 10}
B3 -> {1, 2, 4, 5, 6, 7, 8, 9, 10}
B4 -> {1, 2, 4, 5, 6, 7, 8, 9, 10}

**********OUT**********

B1 -> {1, 2}
B2 -> {1, 2, 4, 5, 6, 7, 8, 9, 10}
B3 -> {4, 5, 6, 7, 8, 9, 10}
B4 -> {1, 2, 4, 5, 6, 7, 8, 9, 10}
```