

07-Apr-22

Experiment 11 - Intermediate code generation- Quadruple, Triple, Indirect Triple

Dhawal Patil
RA1911003010575
CSE A2

Aim:

A program to implement Intermediate code generation – Quadruple, Triple, Indirect Triple.

Algorithm:

The algorithm takes a sequence of three-address statements as input. For each three address statements of the form $a := b \text{ op } c$ perform the various actions. These are as follows:

1. Invoke a function getreg to find out the location L where the result of computation $b \text{ op } c$ should be stored.
2. Consult the address description for y to determine y'. If the value of y currently in memory and register both then prefer the register y'. If the value of y is not already in L then generate the instruction $\text{MOV } y', L$ to place a copy of y in L.
3. Generate the instruction $\text{OP } z', L$ where z' is used to show the current location of z. if z is in both then prefer a register to a memory location. Update the address descriptor of x to indicate that x is in location L. If x is in L then update its descriptor and remove x from all other descriptors.
4. If the current value of y or z have no next uses or not live on exit from the block or in register then alter the register descriptor to indicate that after execution of $x := y \text{ op } z$ those register will no longer contain y or z.

Code:

```
OPERATORS = set(['+', '-', '*', '/', '(', ')'])
PRI = {'+': 1, '-': 1, '*': 2, '/': 2}
```

```
def infix_to_postfix(formula):
    stack = []
    output = ""
    for ch in formula:
        if ch not in OPERATORS:
            output += ch
        elif ch == '(':
            stack.append('(')
        elif ch == ')':
            while stack and stack[-1] != '(':
                output += stack.pop()
            stack.pop()
        else:
            while stack and stack[-1] != '(' and PRI[ch] <= PRI[stack[-1]]:
                output += stack.pop()
            stack.append(ch)
    while stack:
        output += stack.pop()
```

return output

```
def infix_to_prefix(formula):
    op_stack = []
    exp_stack = []
    for ch in formula:
        if not ch in OPERATORS:
            exp_stack.append(ch)
        elif ch == '(':
            op_stack.append(ch)
        elif ch == ')':
            while op_stack[-1] != '(':
                op = op_stack.pop()
                a = exp_stack.pop()
                b = exp_stack.pop()
                exp_stack.append(op+b+a)
            op_stack.pop()
        else:
            while op_stack and op_stack[-1] != '(' and PRI[ch] <= PRI[op_stack[-1]]:
                op = op_stack.pop()
                a = exp_stack.pop()
                b = exp_stack.pop()
                exp_stack.append(op+b+a)
            op_stack.append(ch)
    while op_stack:
        op = op_stack.pop()
        a = exp_stack.pop()
        b = exp_stack.pop()
        exp_stack.append(op+b+a)
    return exp_stack[-1]
```

```
def generate3AC(pos):
    print("### THREE ADDRESS CODE GENERATION ###")
    exp_stack = []
    t = 1
    for i in pos:
        if i not in OPERATORS:
            exp_stack.append(i)
        else:
            print(f't{t} := {exp_stack[-2]} {i} {exp_stack[-1]}')
            exp_stack = exp_stack[:-2]
            exp_stack.append(f't{t}')
            t += 1
```

```
expres = input("INPUT THE EXPRESSION: ")
pre = infix_to_prefix(expres)
pos = infix_to_postfix(expres)
generate3AC(pos)
```

```
def Quadruple(pos):
    stack = []
    op = []
    x = 1
    for i in pos:
        if i not in OPERATORS:
            stack.append(i)
        elif i == '-':
```

```

op1 = stack.pop()
stack.append("t(%s)" % x)
print("{0:^4s} | {1:^4s} | {2:^4s}|{3:4s}".format(
    i, op1, "(-)", " t(%s)" % x))
x = x+1
if stack != []:
    op2 = stack.pop()
    op1 = stack.pop()
    print("{0:^4s} | {1:^4s} | {2:^4s}|{3:4s}".format(
        "+", op1, op2, " t(%s)" % x))
    stack.append("t(%s)" % x)
    x = x+1
elif i == '=':
    op2 = stack.pop()
    op1 = stack.pop()
    print("{0:^4s} | {1:^4s} | {2:^4s}|{3:4s}".format(i, op2, "(-)", op1))
else:
    op1 = stack.pop()
    op2 = stack.pop()
    print("{0:^4s} | {1:^4s} | {2:^4s}|{3:4s}".format(
        i, op2, op1, " t(%s)" % x))
    stack.append("t(%s)" % x)
    x = x+1

```

```

def Triple(pos):
    stack = []
    op = []
    x = 0
    for i in pos:
        if i not in OPERATORS:
            stack.append(i)
        elif i == '-':
            op1 = stack.pop()
            stack.append("(%s)" % x)
            print("{0:^4s} | {1:^4s} | {2:^4s}".format(i, op1, "(-)"))
            x = x+1
        if stack != []:
            op2 = stack.pop()
            op1 = stack.pop()
            print("{0:^4s} | {1:^4s} | {2:^4s}".format("+", op1, op2))
            stack.append("(%s)" % x)
            x = x+1
        elif i == '=':
            op2 = stack.pop()
            op1 = stack.pop()
            print("{0:^4s} | {1:^4s} | {2:^4s}".format(i, op1, op2))
        else:
            op1 = stack.pop()
            if stack != []:
                op2 = stack.pop()
                print("{0:^4s} | {1:^4s} | {2:^4s}".format(i, op2, op1))
                stack.append("(%s)" % x)
                x = x+1

```

```

def IndirectTriple(pos):
    stack = []
    op = []

```

```

x = 0
c = 0
for i in pos:
    if i not in OPERATORS:
        stack.append(i)
    elif i == '-':
        op1 = stack.pop()
        stack.append("(%s)" % x)
        print("{0:^4s} | {1:^4s} | {2:^4s} | {3:^5d}".format(i, op1, "(-)", c))
        x = x+1
    if stack != []:
        op2 = stack.pop()
        op1 = stack.pop()
        print("{0:^4s} | {1:^4s} | {2:^4s} | {3:^5d}".format(
            "+", op1, op2, c))
        stack.append("(%s)" % x)
        x = x+1
        c = c+1
    elif i == '=':
        op2 = stack.pop()
        op1 = stack.pop()
        print("{0:^4s} | {1:^4s} | {2:^4s} | {3:^5d}".format(i, op1, op2, c))
        c = c+1
    else:
        op1 = stack.pop()
        if stack != []:
            op2 = stack.pop()
            print("{0:^4s} | {1:^4s} | {2:^4s} | {3:^5d}".format(
                i, op2, op1, c))
            stack.append("(%s)" % x)
            x = x+1
            c = c+1
z = 35
print("Statement|Location")
for i in range(0, c):
    print("{0:^4d} | {1:^4d}".format(i, z))
    z = z+1

print("====Quadruple====")
print("Op | Src1 | Src2| Res")
Quadruple(pos)
print("====Tripple====")
print("Op | Src1 | Src2")
Triple(pos)
print("====Indirect Tripple====")
print("Op | Src1 | Src2 |Statement")
IndirectTriple(pos)

```

main.py

```
1 OPERATORS = set(['+', '-', '*', '/', '(', ')'])
2 PRI = {'+': 1, '-': 1, '*': 2, '/': 2}
3
4 def infix_to_postfix(formula):
5     stack = []
6     output = ''
7     for ch in formula:
8         if ch not in OPERATORS:
9             output += ch
10        elif ch == '(':
11            stack.append('(')
12        elif ch == ')':
13            while stack and stack[-1] != '(':
14                output += stack.pop()
15            stack.pop()
16        else:
17            while stack and stack[-1] != '(' and PRI[ch] <= PRI[stack[-1]]:
18                output += stack.pop()
19            stack.append(ch)
20    while stack:
21        output += stack.pop()
22    return output
23
24 def infix_to_prefix(formula):
25     op_stack = []
26     exp_stack = []
27     for ch in formula:
28         if not ch in OPERATORS:
29             exp_stack.append(ch)
30         elif ch == '(':
31             op_stack.append(ch)
32         elif ch == ')':
33             while op_stack[-1] != '(':
34                 op = op_stack.pop()
35                 a = exp_stack.pop()
36                 b = exp_stack.pop()
37                 exp_stack.append(op+b+a)
38             op_stack.pop()
39         else:
40             while op_stack and op_stack[-1] != '(' and PRI[ch] <= PRI[op_stack[-1]]:
41                 op = op_stack.pop()
42                 a = exp_stack.pop()
43                 b = exp_stack.pop()
44                 exp_stack.append(op+b+a)
45             op_stack.append(ch)
46     while op_stack:
47         op = op_stack.pop()
48         a = exp_stack.pop()
49         b = exp_stack.pop()
50         exp_stack.append(op+b+a)
51     return exp_stack[-1]
52
53 def generate3AC(pos):
54     print("### THREE ADDRESS CODE GENERATION ###")
55     exp_stack = []
56     t = 1
57     for i in pos:
58         if i not in OPERATORS:
59             exp_stack.append(i)
60         else:
61             print(f't{t} := {exp_stack[-2]} {i} {exp_stack[-1]}')
62             exp_stack = exp_stack[:-2]
63             exp_stack.append(f't{t}')
64             t += 1
65
66 expres = input("INPUT THE EXPRESSION: ")
67 pre = infix_to_prefix(expres)
68 pos = infix_to_postfix(expres)
69 generate3AC(pos)
70
71 def Quadruple(pos):
72     stack = []
73     op = []
```

```

74 x = 1
75 for i in pos:
76     if i not in OPERATORS:
77         stack.append(i)
78     elif i == '-':
79         op1 = stack.pop()
80         stack.append("t(%s)" % x)
81         print("{0:^4s} | {1:^4s} | {2:^4s}|{3:4s}".format(
82             i, op1, "(-)", " t(%s)" % x))
83         x = x+1
84     if stack != []:
85         op2 = stack.pop()
86         op1 = stack.pop()
87         print("{0:^4s} | {1:^4s} | {2:^4s}|{3:4s}".format(
88             "+", op1, op2, " t(%s)" % x))
89         stack.append("t(%s)" % x)
90         x = x+1
91     elif i == '=':
92         op2 = stack.pop()
93         op1 = stack.pop()
94         print("{0:^4s} | {1:^4s} | {2:^4s}|{3:4s}".format(i, op2, "(-)", op1))
95     else:
96         op1 = stack.pop()
97         op2 = stack.pop()
98         print("{0:^4s} | {1:^4s} | {2:^4s}|{3:4s}".format(
99             i, op2, op1, " t(%s)" % x))
100         stack.append("t(%s)" % x)
101         x = x+1
102
103 def Triple(pos):
104     stack = []
105     op = []
106     x = 0
107     for i in pos:
108         if i not in OPERATORS:
109             stack.append(i)
110
111         elif i == '-':
112             op1 = stack.pop()
113             stack.append("(%s)" % x)
114             print("{0:^4s} | {1:^4s} | {2:^4s}".format(i, op1, "(-)"))
115             x = x+1
116         if stack != []:
117             op2 = stack.pop()
118             op1 = stack.pop()
119             print("{0:^4s} | {1:^4s} | {2:^4s}".format("+", op1, op2))
120             stack.append("(%s)" % x)
121             x = x+1
122         elif i == '=':
123             op2 = stack.pop()
124             op1 = stack.pop()
125             print("{0:^4s} | {1:^4s} | {2:^4s}".format(i, op1, op2))
126         else:
127             op1 = stack.pop()
128             if stack != []:
129                 op2 = stack.pop()
130                 print("{0:^4s} | {1:^4s} | {2:^4s}".format(i, op2, op1))
131                 stack.append("(%s)" % x)
132                 x = x+1
133
134 def IndirectTriple(pos):
135     stack = []
136     op = []
137     x = 0
138     c = 0
139     for i in pos:
140         if i not in OPERATORS:
141             stack.append(i)
142         elif i == '-':
143             op1 = stack.pop()
144             stack.append("(%s)" % x)
145             print("{0:^4s} | {1:^4s} | {2:^4s} | {3:^5d}".format(i, op1, "(-)", c))
146             x = x+1
147         if stack != []:
148             op2 = stack.pop()
149             op1 = stack.pop()
150             print("{0:^4s} | {1:^4s} | {2:^4s} | {3:^5d}".format(
151                 "+", op1, op2, c))
152             stack.append("(%s)" % x)
153             x = x+1
154             c = c+1
155         elif i == '=':
156             op2 = stack.pop()
157             op1 = stack.pop()
158             print("{0:^4s} | {1:^4s} | {2:^4s} | {3:^5d}".format(i, op1, op2, c))
159             c = c+1
160         else:
161             op1 = stack.pop()

```

```

161-         if stack != []:
162-             op2 = stack.pop()
163-             print("{0:^4s} | {1:^4s} | {2:^4s} | {3:^5d}".format(
164-                 i, op2, op1, c))
165-             stack.append("(%s)" % x)
166-             x = x+1
167-             c = c+1
168-
169-         z = 35
170-         print("Statement|Location")
171-         for i in range(0, c):
172-             print("{0:^4d} | {1:^4d}".format(i, z))
173-             z = z+1
174-
175-         print("====Quadruple====")
176-         print("Op      | Src1 | Src2| Res")
177-         Quadruple(pos)
178-         print("====Tripple====")
179-         print("Op      | Src1 | Src2")
180-         Triple(pos)
181-         print("====Indirect Tripple====")
182-         print("Op      | Src1 | Src2 |Statement")
183-         IndirectTriple(pos)

```

Output:

```

INPUT THE EXPRESSION: (A+B)*(A+B+C)
### THREE ADDRESS CODE GENERATION ###
t1 := A + B
t2 := A + B
t3 := t2 + C
t4 := t1 * t3
====Quadruple====
Op      | Src1 | Src2| Res
+       | A    | B   | t(1)
+       | A    | B   | t(2)
+       | t(2) | C   | t(3)
*       | t(1) | t(3) | t(4)
====Tripple====
Op      | Src1 | Src2
+       | A    | B
+       | A    | B
+       | (1)  | C
*       | (0)  | (2)
====Indirect Tripple====
Op      | Src1 | Src2 |Statement
+       | A    | B    | 0
+       | A    | B    | 1
+       | (1)  | C    | 2
*       | (0)  | (2)  | 3
Statement|Location
0        | 35
1        | 36
2        | 37
3        | 38

...Program finished with exit code 0
Press ENTER to exit console.

```

Result:

The program was successfully compiled and run.