University of Asia Pacific
Department of Computer Science & Engineering

Course Title: Compiler Design Lab
Course Code: CSE 430
Lab 5 Report

Submitted by:-
Junnatul Mawa
Section:-B1
Registration:- 20101070

Three Address Code Generation

Three-address code (often abbreviated to TAC or 3AC) is an intermediate code used by
optimizing compilers to aid in the implementation of code-improving transformations. Three
address code is easy to generate and can be easily converted to machine code. It makes use of
at most three addresses and one operator to represent an expression and the value computed
at each instruction is stored in a temporary variable generated by the compiler. The compiler
decides the order of operation givenby three address code.

General representation of TAC:

a = b op c

Where a, b, or c represents operands like names, constants, or compiler-generated temporariesand op represents the operator.
Example:
x = (-b + sqrt(b^2 - 4*a*c)) / (2*a)
TAC of Example:

t1 := b * b
t2 := 4 * a
t3 := t2 * c
t4 := t1 - t3
t5 := sqrt(t4)
t6 := 0 - b
t7 := t5 + t6
t8 := 2 * a
t9 := t7 / t8
x := t9

Implementation Details:
Precedence:
While implementing the TAC, the first thing to remember is precedence.
Precedence/
order of operations is a collection of rules that reflect conventions about which procedures to perform first to evaluate a given mathematical expression.
The precedence of an operator specifies how "tightly" it binds two expressions together. For example, in the expression 1 + 5 * 3, the answer is 16 and not 18 because the multiplication
("*")operator has higher precedence than the addition ("+") operator. Parentheses may be used toforce precedence, if necessary. For instance: (1 + 5) * 3 evaluates to 18.

Here is the code:-

```
            while stack and stack[-1] != "(":
                temp_stack.append(stack.pop())
            if stack and stack[-1] == "(":
                stack.pop()
            three_address_code.append((temp_counter, temp_stack[::-1]))
            stack.append("t" + str(temp_counter))
            temp_counter += 1
        else:
            stack.append(char)

    three_address_code.append((temp_counter, stack))

    result_queue = deque()
    temp_mapping = defaultdict(int)
    temp_counter = 1

    # Generate three-address code
    while three_address_code:
        current_entry = three_address_code.popleft()
        index = current_entry[0]
        current_exp = current_entry[1]

        # Update temporary variable mapping
        for i in range(len(current_exp)):
            if current_exp[i][0] == 't':
                current_exp[i] = 't' + str(temp_mapping[int(current_exp[i][1:])])

        while len(current_exp) > 2:
```

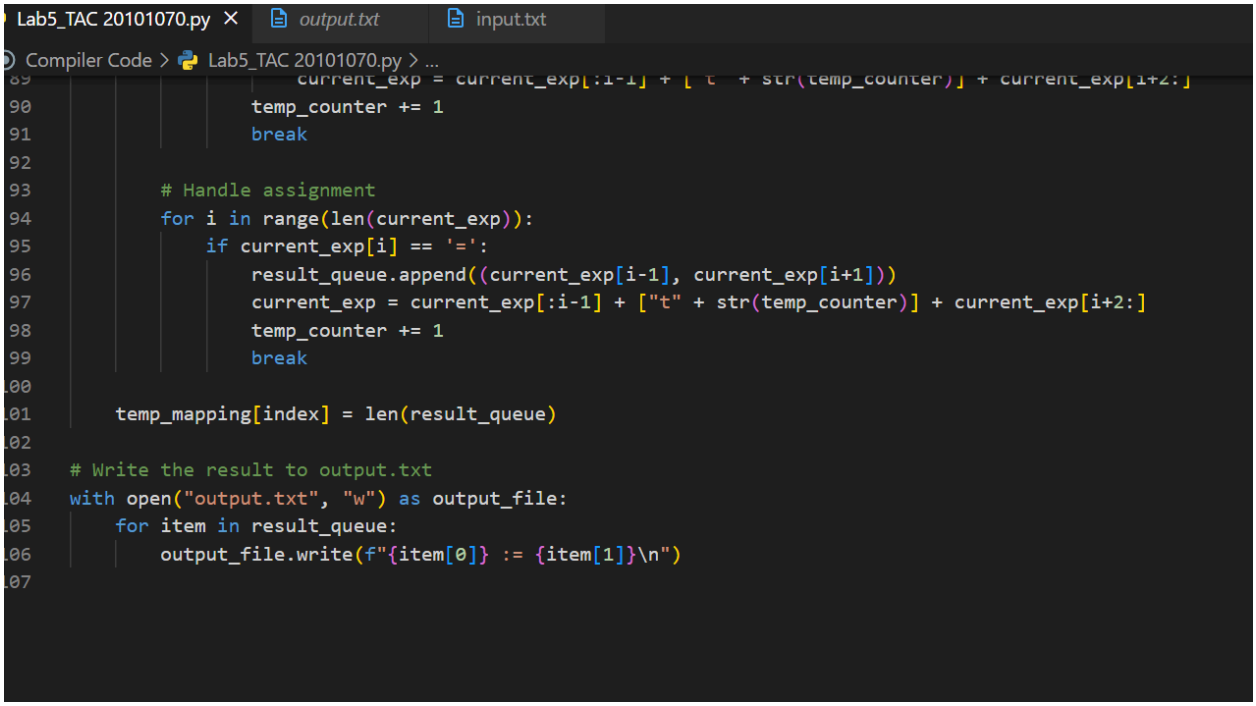
Compiler Code > Lab5_TAC 20101070.py > ...

```
43          if current_exp[i][0] == 't':
44              current_exp[i] = 't' + str(temp_mapping[int(current_exp[i][1:])])
45
46      while len(current_exp) > 2:
47          # Handle sqrt operator
48          for i in range(len(current_exp)):
49              if current_exp[i] == 'sqrt':
50                  result_queue.append(("t" + str(temp_counter), "sqrt(" + current_exp[i+1] + ")"))
51                  current_exp = current_exp[:i] + ["t" + str(temp_counter)] + current_exp[i+2:]
52                  temp_counter += 1
53                  break
54
55          # Handle exponentiation
56          for i in range(len(current_exp)):
57              if current_exp[i] == '^':
58                  temp_exp = deque("*".join([current_exp[i-1]] * int(current_exp[i+1])))
59                  while len(temp_exp) > 2:
60                      for j in range(len(temp_exp)):
61                          if temp_exp[j] == '*':
62                              result_queue.append(("t" + str(temp_counter), temp_exp[j-1] + "*" + temp_exp[j+1]))
63                              temp_exp = deque([*temp_exp[:j-1], "t" + str(temp_counter), *temp_exp[j+2:]])
64                              temp_counter += 1
65                              break
66                  current_exp = ["t" + str(temp_counter - 1)] + current_exp[i+2:]
67                  break
68
69          # Handle multiplication, division, and modulus
70          flag = False
71          for i in range(len(current_exp)):
```

Lab5_TAC 20101070.py | output.txt | input.txt

Compiler Code > Lab5_TAC 20101070.py > ...

```
72              if current_exp[i] in ['*', '/', '%']:
73                  result_queue.append(("t" + str(temp_counter), current_exp[i-1] + current_exp[i] + current_exp[i+1]))
74                  current_exp = current_exp[:i-1] + ["t" + str(temp_counter)] + current_exp[i+2:]
75                  temp_counter += 1
76                  flag = True
77                  break
78          if flag:
79              continue
80
81          # Handle addition and subtraction
82          for i in range(len(current_exp)):
83              if current_exp[i] in ['+', '-']:
84                  if current_exp[i] == '-' and i == 0:
85                      result_queue.append(("t" + str(temp_counter), '0' + current_exp[i] + current_exp[i+1]))
86                      current_exp = ["t" + str(temp_counter)] + current_exp[i+2:]
87                  else:
88                      result_queue.append(("t" + str(temp_counter), current_exp[i-1] + current_exp[i] + current_exp[i+1
89                      current_exp = current_exp[:i-1] + ["t" + str(temp_counter)] + current_exp[i+2:]
90                  temp_counter += 1
91                  break
92
93          # Handle assignment
94          for i in range(len(current_exp)):
95              if current_exp[i] == '=':
96                  result_queue.append((current_exp[i-1], current_exp[i+1]))
97                  current_exp = current_exp[:i-1] + ["t" + str(temp_counter)] + current_exp[i+2:]
98                  temp_counter += 1
99                  break
```



Input.txt

```
void main()
{
int v, l, c;
//comment
int v = l*c + 10;
}
```