BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Empowered Autonomous Institute Affiliated to University of Mumbai)
[Knowledge is Nectar]

Department of Computer Engineering

Course - System Programming and Compiler Construction (SPCC)

UID	2021300108
Name	Hatim Sawai
Class and Batch	TE Computer Engineering Class B – Batch C
Date	25/3/2024
Lab #	5
Aim	Intermediate Code Generation
Objective	Input: Program should accept a expression in Postfix format Output: Should be in a table format shown below known as Quadraple format. (make use of Data Structure stack)
Theory	**Intermediate Code Generation (ICG):** ICG is a crucial stage in compiler design that bridges the gap between the source code and the target machine code. It involves translating the parsed and analyzed source code representation into an intermediate representation, known as intermediate code (IC). This IC is: Machine-independent: It is not specific to any particular target machine architecture, allowing the same IC to be used for different platforms. Easier to optimize: The IC offers a higher-level abstraction compared to machine code, making it more convenient to apply various optimization techniques. Facilitates code generation: Different target machines can have their own backends that translate the IC into their specific machine code. **Common Types of Intermediate Code:** Three-Address Code (TAC): A common format where each statement is represented by an assignment involving three operands (e.g., a = b + c). Syntax Tree: A tree-like structure representing the program's syntax, where each node corresponds to a construct or expression in the source code. Postfix Notation: A linear representation where operands are listed before the operator (e.g., b c + for b + c). **Data structures for Three-Address codes** 1. Quadruples Has four fields: op, arg1, arg2 and result 2. Triples Temporaries are not used and instead references to instructions are made 3. Indirect triples In addition to triples we use a list of pointers to triples

Implementation / Code

```
def precedence(op):
    if op in ["+", "-"]:
        return 1
    if op in ["*", "/"]:
        return 2
    if op == "^":
        return 3
    return 0

def infic_to_postfix(exp):
    stack = []
    postfix = ""
    for char in exp:
        if char.isalnum():
            postfix += char
        elif char == "(":
            stack.append(char)
        elif char == ")":
            while stack and stack[-1] != "(":
                postfix += stack.pop()
            stack.pop()
        else:
            while stack and stack[-1] != "(" and precedence(stack[-1]) >=
precedence(char):
                postfix += stack.pop()
            stack.append(char)
    while stack:
        postfix += stack.pop()
    return postfix

def is_op(char):
    return char in ["+", "-", "*", "/", "^", "="]

def display_quadruple(quadruples):
    print("Quadruple Representation:")
    print("{:<10} {:<10} {:<10} {:<10}".format("Operator", "Arg1", "Arg2",
"Result"))
    for quadruple in quadruples:
        print("{:<10} {:<10} {:<10} {:<10}".format(*quadruple))

def postfix_to_quadruple(exp):
```

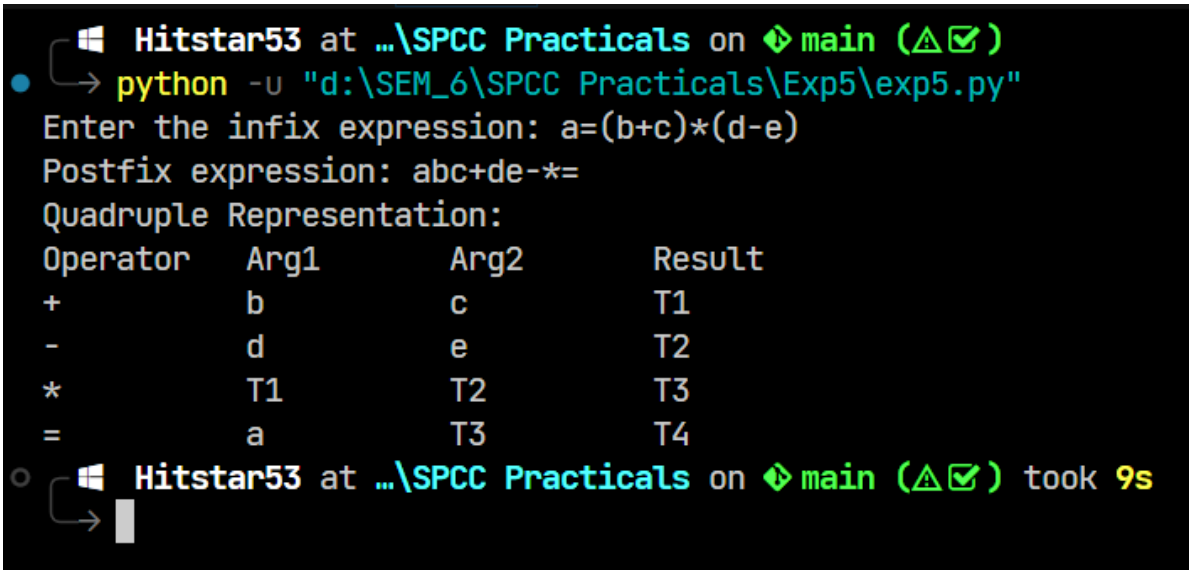
```
    stack = []
    quadruples = []
    temp = 1
    for char in exp:
        if char.isalnum():
            stack.append(char)
        elif is_op(char):
            op2 = stack.pop()
            op1 = stack.pop()
            temp_var = "T" + str(temp)
            temp += 1
            quadruples.append([char, op1, op2, temp_var])
            stack.append(temp_var)
    return quadruples


def main():
    exp = input("Enter the infix expression: ")
    exp = infic_to_postfix(exp)
    print("Postfix expression:", exp)
    quadruples = postfix_to_quadruple(exp)
    display_quadruple(quadruples)


if __name__ == "__main__":
    main()
```

BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Empowered Autonomous Institute Affiliated to University of Mumbai)
[Knowledge is Nectar]

Department of Computer Engineering

Output	
Conclusion	In this experiment, we learned how to convert postfix expressions, a high-level notation, into quadruples, a form of intermediate code. This approach utilizes a stack to process the expression and generates a table representing each operation as a quadruple.
References	Rajmane Sir – PPT: Unit-3.1 Compiler Design_Code Generation.pdf