```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MAX_SIZE 100

char stack[MAX_SIZE];
int top = -1;

typedef struct {
    char code[MAX_SIZE][MAX_SIZE];
    int count;
} ThreeAddressCode;

void emit(ThreeAddressCode *code, const char *op, const char *arg1, const char *arg2,
const char *result)
{
    sprintf(code->code[code->count], "%c = %c %c %c", result[0], arg1[0], op[0], arg2[0]);
    code->count++;
}


void push(char operator)
{
    if (top == MAX_SIZE - 1)
    {
        printf("\nStack overflow");
        exit(1);
    }
    else
    {
        top++;
        stack[top] = operator;
    }
}

char pop()
{
    if (top == -1)
        return -1;

    char ch = stack[top];
    top--;
    return ch;
}

int is_operator(char symbol)
{
    return (symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' || symbol ==
'-');
}

int precedence(char symbol)
{
    if (symbol == '^')
        return 3;
    else if (symbol == '*' || symbol == '/')
        return 2;
    else if (symbol == '+' || symbol == '-')
        return 1;
    else
        return 0;
}

void infixToPostfix(char infix_exp[], char postfix_exp[])
{
    int i, j;
    char x, item;

    push('(');
    strcat(infix_exp, ")");
    i = 0;
    j = 0;
    item = infix_exp[i];

    while (item != '\0')
    {
```

```
        if (item == '(')
        {
            push(item);
        }
        else if (isdigit(item) || isalpha(item))
        {
            postfix_exp[j] = item;
            j++;
        }
        else if (is_operator(item))
        {
            x = pop();
            while (is_operator(x) && precedence(x) >= precedence(item))
            {
                postfix_exp[j] = x;
                j++;
                x = pop();
            }
            push(x);
            push(item);
        }
        else if (item == ')')
        {
            x = pop();
            while (x != '(')
            {
                postfix_exp[j] = x;
                j++;
                x = pop();
            }
        }
        else
        {
            printf("\nInvalid infix Expression.\n");
            getchar();
            exit(1);
        }
        i++;
        item = infix_exp[i];
    }

    postfix_exp[j] = '\0';
}

// Declaration of the isEmpty function
int isEmpty()
{
    return top == -1;
}

void postfixToThreeAddressCode(char postfix_exp[], ThreeAddressCode *code)
{
    int i = 0;
    char operand[MAX_SIZE];
    int tempCount = 0;

    while (postfix_exp[i] != '\0')
    {
        if (isdigit(postfix_exp[i]) || isalpha(postfix_exp[i]))
        {
            snprintf(operand, sizeof(operand), "%c", postfix_exp[i]);
            push(operand[0]);
        }
        else if (is_operator(postfix_exp[i]))
        {
            char arg2 = pop();
            char arg1 = pop();
            char result[3];  // Temporary variable (e.g., A, B, C, ...)
            snprintf(result, sizeof(result), "%c", 'A' + tempCount++);
            emit(code, &postfix_exp[i], &arg1, &arg2, result);
            push(result[0]);
        }
        i++;
    }

    // Ensure there is only one result left in the stack
    char finalResult = pop();
    if (finalResult == -1)
```

```
    {
        printf("\nInvalid postfix Expression.\n");
        exit(1);
    }

    // If there are any remaining items in the stack, the postfix expression is invalid
    if (pop() != -1)
    {
        printf("\nInvalid postfix Expression.\n");
        exit(1);
    }
}

// Function to print Quadruple in Table format
void printQuadrupleTable(ThreeAddressCode *code)
{
    printf("\nQuadruple Format:\n");
    printf("| %-6s | %-10s | %-10s | %-10s | %-10s |\n", "LineNo", "Operator",
"Argument1", "Argument2", "Result");
    printf("|--------|------------|------------|------------|------------|\n");

    for (int i = 0; i < code->count; i++)
    {
        printf("| %-6d | %-10c | %-10c | %-10c | %-10c |\n", i, code->code[i][6], code-
>code[i][4], code->code[i][8], code->code[i][0]);
    }
}

// Function to print Triple in Table format
void printTripleTable(ThreeAddressCode *code)
{
    printf("\nTriple Format:\n");
    printf("| %-6s | %-10s | %-10s | %-10s |\n", "LineNo", "Operator", "Argument1",
"Argument2");
    printf("|--------|------------|------------|------------|\n");

    for (int i = 0; i < code->count; i++)
    {
        printf("| %-6d | %-10c | %-10c | %-10c |\n", i, code->code[i][6], code->code[i]
[4], code->code[i][8]);
    }
}

// Function to print Indirect Triple in Table format
void printIndirectTripleTable(ThreeAddressCode *code)
{
    printf("\nIndirect Triple Format:\n");
    printf("| %-6s | %-15s |\n", "LineNo", "Pointer");
    printf("|--------|-----------------|\n");

    int pointerCounter = 14;

    for (int i = 0; i < code->count; i++)
    {
        printf("| %-6d | %-15d |\n", i, pointerCounter++);
    }
}

int main()
{
    char infix[MAX_SIZE], postfix[MAX_SIZE];
    ThreeAddressCode code;
    code.count = 0;

    printf("Enter Infix expression: ");
    scanf("%s", infix);

    infixToPostfix(infix, postfix);
    printf("Postfix Expression: %s\n", postfix);

    postfixToThreeAddressCode(postfix, &code);

    printf("\nGenerated Three-Address Code:\n");
    for (int i = 0; i < code.count; i++)
    {
        printf("%s\n", code.code[i]);
    }
```

```
    printQuadrupleTable(&code);
    printTripleTable(&code);
    printIndirectTripleTable(&code);

    return 0;
}
```