The aim of this program is to evaluate a postfix expression provided by the user and output the calculated result.

Aim

The aim of this program is to evaluate a postfix expression provided by the user and output the calculated result.

Theory

- **Postfix Expressions:** Postfix notation, also known as Reverse Polish Notation (RPN), is a way of writing mathematical expressions where the operators follow the operands. This eliminates the need for parentheses and simplifies the evaluation process using a stack.
- **Stack Data Structure:** A stack is a Last-In-First-Out (LIFO) data structure. We'll use a stack to temporarily store operands as we encounter them in the postfix expression. When an operator is found, we pop the required number of operands from the stack, perform the operation, and push the result back onto the stack.

Procedure

1. **Read the Postfix Expression:**
   - Prompt the user to enter a postfix expression.
   - Use `fgets` to safely read the expression into a character array.
2. **Initialize the Stack:**
   - Create an array to represent the stack and set the `top` index to -1 (indicating an empty stack).
3. **Evaluate the Expression:**
   - Iterate through each character in the postfix expression.
   - If the character is an operand (digit):
     - Convert it to a numeric value.
     - Push it onto the stack.
   - If the character is an operator:
     - Check if there are enough operands on the stack (at least two).
     - Pop the top two operands from the stack.
     - Perform the operation using the `compute` function.
     - Push the result back onto the stack.
   - Ignore whitespace characters.
4. **Get the Result:**

- After processing the entire expression, the final result will be the only element remaining on the stack.
- Pop the result from the stack.

5. **Print the Result:**
   - Display the calculated result to the user.
6. **Error Handling:**
   - Include checks for invalid postfix expressions (e.g., not enough operands for an operator, invalid operators).
   - Handle division by zero gracefully.

Code (Provided in the prompt)

C

```
#include <stdio.h>
#include <ctype.h>  // For isdigit()
#include <math.h>   // For pow()

#define MAX_SIZE 20

int main() {
    double stack[MAX_SIZE];
    int top = -1;
    char postfix[MAX_SIZE], symbol;
    double op1, op2, result;

    // Get postfix expression from user
    printf("Enter the postfix expression: ");
    fgets(postfix, MAX_SIZE, stdin); // Use fgets for input safety

    // Evaluate postfix expression
    for (int i = 0; postfix[i] != '\0' && postfix[i] != '\n'; i++) {
        symbol = postfix[i];

        if (isdigit(symbol)) {
            // Push operand onto the stack
            stack[++top] = symbol - '0'; // Convert char digit to integer
        } else if (isspace(symbol)) {
            continue; // Ignore whitespace
        } else {
            // Operator encountered, pop two operands and evaluate
            if (top < 1) {
                printf("Error: Invalid postfix expression!\n");
```

```
                return 1; // Indicate an error occurred
            }
            op2 = stack[top--];
            op1 = stack[top--];

            switch (symbol) {
                case '+': result = op1 + op2; break;
                case '-': result = op1 - op2; break;
                case '*': result = op1 * op2; break;
                case '/':
                    if (op2 == 0) {
                        printf("Error: Division by zero!\n");
                        return 1;
                    }
                    result = op1 / op2;
                    break;
                case '^': result = pow(op1, op2); break;
                default:
                    printf("Error: Invalid operator '%c'!\n", symbol);
                    return 1;
            }

            stack[++top] = result; // Push result back onto the stack
        }
    }

    // Check if the expression was valid
    if (top != 0) {
        printf("Error: Invalid postfix expression!\n");
        return 1;
    }

    printf("The result is: %f\n", stack[0]);
    return 0;
}
```

Learning Outcomes

- Understanding postfix notation and its advantages
- Implementing stack operations in C
- Applying a stack to solve a real-world problem (postfix evaluation)
- Error handling in C programs

Feel free to ask if you would like any modifications or additional features added to this code!