**Mini Project Report: Infix to Postfix Conversion and Evaluation Using C**

**ABSTRACT:**

This project focuses on designing and implementing a C-based application to convert an infix expression to postfix (also known as Reverse Polish Notation) and evaluate the resultant postfix expression. The primary aim is to simplify arithmetic expression evaluation using data structures like stacks. The application demonstrates the power and utility of stack data structures in expression conversion and evaluation, which are widely used in compilers and calculators.

**ACKNOWLEDGEMENT**

I would like to express my sincere gratitude to my Data Structures faculty and lab instructors for their continuous support, guidance, and motivation throughout the completion of this mini project. I would also like to thank my classmates and friends for their valuable feedback and encouragement.

**Introduction**

In programming and mathematical computation, evaluating complex expressions is a common task. Infix expressions, where operators are placed between operands, are not the most efficient format for computation. Postfix expressions are more straightforward to evaluate programmatically. This project implements both the conversion from infix to postfix and the evaluation of postfix expressions using stacks.

**Problem Definition**

1. To create a C program that:
2. Converts a given infix expression into a postfix expression.
3. Evaluates the resultant postfix expression.
4. Displays the result to the user.

**Objective**

* To understand the use of stack data structures in expression evaluation.
* To implement infix to postfix conversion algorithm.
* To evaluate postfix expressions accurately.
* To provide a user-friendly application demonstrating these functionalities.

**System Design**

* **Input**: Infix expression (e.g., (3+4)\*5)
* **Output**: Postfix expression (e.g., 3 4 + 5 \*) and its evaluated result (35)
* **Data Structure Used**: Stack for operators during conversion and for operands during evaluation

**Flowchart**

**START**

**│**

**├─[Infix Conversion Phase]───────────────────────────────────┐**

**│ │**

**▼ │**

**Read Infix Expression │**

**Initialize Operator Stack & Postfix String │**

**│ │**

**▼ │**

**For each character in infix expression: │**

**│ Is it an operand (A-Z/a-z/0-9)? │**

**│ ├─Yes─► Add to Postfix String │**

**│ │ │**

**│ ├─No─► Is it '('? │**

**│ │ ├─Yes─► Push to Stack │**

**│ │ │ │**

**│ │ ├─No─► Is it ')'? │**

**│ │ │ ├─Yes─► Pop from Stack to Postfix until '(' │**

**│ │ │ │ (Discard both parentheses) │**

**│ │ │ │ │**

**│ │ │ ├─No─► Operator: │**

**│ │ │ While stack not empty AND │**

**│ │ │ (stack.top != '(' AND precedence ≤ current): │**

**│ │ │ Pop operator to Postfix │**

**│ │ │ Push current operator │**

**│ ▼ ▼ │**

**├─End of Infix Expression─► Pop all operators to Postfix │**

**│ │**

**└─► Postfix Expression Generated │**

**│ │**

**├─[Postfix Evaluation Phase]─────────────────────────────────┘**

**│**

**▼**

**Read Postfix Expression**

**Initialize Operand Stack**

**│**

**▼**

**For each token in postfix:**

**│ Is it an operand?**

**│ ├─Yes─► Push to Stack**

**│ │**

**│ ├─No─► Operator:**

**│ Pop operand2**

**│ Pop operand1**

**│ Compute operand1 [operator] operand2**

**│ Push result back to Stack**

**│**

**▼**

**After processing all tokens:**

**│ Result = Stack Top**

**│**

**▼**

**Display Final Result**

**│**

**END**

**Source Code**

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#define MAX 100

char stack[MAX];

int top = -1;

void push(char c) {

stack[++top] = c;

}

char pop() {

return stack[top--];

}

char peek() {

return stack[top];

}

int precedence(char op) {

switch(op) {

case '+':

case '-': return 1;

case '\*':

case '/': return 2;

case '^': return 3;

default: return 0;

}

}

int isOperator(char c) {

return (c == '+' || c == '-' || c == '\*' || c == '/' || c == '^');

}

void infixToPostfix(char\* infix, char\* postfix) {

int i = 0, k = 0;

char c;

while((c = infix[i++]) != '\0') {

if(isdigit(c)) {

postfix[k++] = c;

} else if(c == '(') {

push(c);

} else if(c == ')') {

while(top != -1 && peek() != '(') {

postfix[k++] = pop();

}

pop();

} else if(isOperator(c)) {

while(top != -1 && precedence(peek()) >= precedence(c)) {

postfix[k++] = pop();

}

push(c);

}

}

while(top != -1) {

postfix[k++] = pop();

}

postfix[k] = '\0';

}

int evaluatePostfix(char\* postfix) {

int i = 0;

int evalStack[MAX];

int evalTop = -1;

char c;

while((c = postfix[i++]) != '\0') {

if(isdigit(c)) {

evalStack[++evalTop] = c - '0';

} else {

int b = evalStack[evalTop--];

int a = evalStack[evalTop--];

switch(c) {

case '+': evalStack[++evalTop] = a + b; break;

case '-': evalStack[++evalTop] = a - b; break;

case '\*': evalStack[++evalTop] = a \* b; break;

case '/': evalStack[++evalTop] = a / b; break;

case '^': {

int res = 1;

for(int j = 0; j < b; j++) res \*= a;

evalStack[++evalTop] = res;

break;

}

}

}

}

return evalStack[evalTop];

}

int main() {

char infix[MAX], postfix[MAX];

printf("Enter infix expression: ");

scanf("%s", infix);

infixToPostfix(infix, postfix);

printf("Postfix expression: %s\n", postfix);

printf("Evaluated result: %d\n", evaluatePostfix(postfix));

return 0;

}

**Outputs** *Input:* (3+4)*5  
Postfix: 34+5*  
*Result:* 35

**Output Overview**

* The program takes infix expressions as input.
* Converts them to postfix using a stack-based algorithm.
* Evaluates the postfix expression using another stack.
* Displays the postfix expression and the final result.

**Challenges**

* Managing stack operations correctly.
* Handling multi-digit numbers (the current version does not support this).
* Properly handling operator precedence and parentheses.

**Future Scope**

* Extend support for multi-digit numbers and floating-point arithmetic.
* Add error handling for malformed expressions.
* Create a GUI-based calculator.
* Optimize algorithm efficiency.

**Conclusion**

This mini project successfully demonstrates how stacks can be used to convert and evaluate expressions. It reinforces the understanding of data structures in real-world problem solving and lays a foundation for more advanced computational tools.

**References**

* "Data Structures Using C" by Reema Thareja
* GeeksforGeeks: Stack Data Structure and Expression Evaluation
* TutorialsPoint: Infix to Postfix Conversion