# Applications of Stack – Expression Conversion & Evaluation in C

## Pseudocode (Sudo Logic)

```
Infix → Postfix
1. For each char in infix:
   - Operand → Output
   - ( → Push
   - ) → Pop until (
   - Operator → Pop higher/equal precedence, then push
2. Pop remaining operators.

Infix → Prefix
1. Reverse infix and swap ( , )
2. Convert to postfix
3. Reverse postfix → prefix

Postfix Evaluation
1. For each token:
   - Operand → Push
   - Operator → Pop 2, apply, push result
2. Top of stack = result

Prefix Evaluation
1. Scan right to left
2. Operand → Push
3. Operator → Pop 2, apply, push
4. Final = result
```

## C Program

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

int isOperatorChar(char c) {
    return c=='+'||c=='-'||c=='*'||c=='/'||c=='^'||c=='%';
}
int precedence(char op) {
    if (op=='^') return 3;
    if (op=='*'||op=='/'||op=='%') return 2;
    if (op=='+'||op=='-') return 1;
    return 0;
}
void reverseStr(char *s) {
    int i=0,j=strlen(s)-1; char t;
    while(i<j){ t=s[i]; s[i]=s[j]; s[j]=t; i++; j--; }
}
void infixToPostfix(const char* infix, char* postfix){
    char stack[256]; int top=-1,k=0;
    for(int i=0; infix[i]; i++){
        char c=infix[i];
        if(isspace(c)) continue;
        if(isalnum(c)) postfix[k++]=c;
        else if(c=='(') stack[++top]=c;
```

```
        else if(c==')'){
            while(top>=0&&stack[top]!='(') postfix[k++]=stack[top--];
            if(top>=0&&stack[top]=='(') top--;
        } else if(isOperatorChar(c)){
            while(top>=0&&isOperatorChar(stack[top])&&precedence(stack[top])>=precedence(c))
                postfix[k++]=stack[top--];
            stack[++top]=c;
        }
    }
    while(top>=0) postfix[k++]=stack[top--];
    postfix[k]='\0';
}
void infixToPrefix(const char* infix,char* prefix){
    int n=strlen(infix),idx=0; char tmp[512];
    for(int i=n-1;i>=0;i--){
        char c=infix[i];
        if(c=='(') c=')'; else if(c==')') c='(';
        tmp[idx++]=c;
    } tmp[idx]='\0';
    char post[512]; infixToPostfix(tmp,post);
    strcpy(prefix,post); reverseStr(prefix);
}
long long applyOpLong(long long a,long long b,char op){
    switch(op){
        case '+':return a+b; case '-':return a-b;
        case '*':return a*b; case '/':return a/b;
        case '%':return a%b;
        case '^':{ long long r=1; for(long long i=0;i<b;i++) r*=a; return r; }
    } return 0;
}
long long evaluatePostfix(const char* expr){
    char* copy=strdup(expr); char* token=strtok(copy," ");
    long long stack[512]; int top=-1;
    while(token){
        if(strlen(token)==1&&isOperatorChar(token[0])){
            long long b=stack[top--],a=stack[top--];
            stack[++top]=applyOpLong(a,b,token[0]);
        } else stack[++top]=atoll(token);
        token=strtok(NULL," ");
    } long long res=stack[top]; free(copy); return res;
}
long long evaluatePrefix(const char* expr){
    char* copy=strdup(expr); char* t=strtok(copy," ");
    char* tokens[128]; int cnt=0;
    while(t){ tokens[cnt++]=strdup(t); t=strtok(NULL," "); }
    long long stack[512]; int top=-1;
    for(int i=cnt-1;i>=0;i--){
        if(strlen(tokens[i])==1&&isOperatorChar(tokens[i][0])){
            long long a=stack[top--],b=stack[top--];
            stack[++top]=applyOpLong(a,b,tokens[i][0]);
        } else stack[++top]=atoll(tokens[i]);
        free(tokens[i]);
    } free(copy); return stack[top];
}
int main(){
    const char* infix="(A+B)*C"; char post[512],pre[512];
    infixToPostfix(infix,post); infixToPrefix(infix,pre);
    printf("Infix : %s\nPostfix : %s\nPrefix : %s\n",infix,post,pre);
    const char* p1="5 3 + 2 *"; printf("Postfix Eval: %lld\n",evaluatePostfix(p1));
    const char* p2="* + 5 3 2"; printf("Prefix Eval: %lld\n",evaluatePrefix(p2));
    return 0;
}
```

## Example Run

```
Infix : (A+B)*C
Postfix : AB+C*
Prefix : *+ABC

Postfix Eval: 16  (for 5 3 + 2 *)
Prefix Eval: 16   (for * + 5 3 2)
```