Infix to Postfix Conversion Algorithm

Step-by-step explanation

Steps for Postfix algorithm

Step1 : Initialization

• Take an empty stack, a postfix array, and the infix expression.

Step 2: Scan the Infix Expression

• Scan the infix expression from left to right.

Step 3: Handle Different Cases

- Case 1: Operand (letter or digit) → Push to postfix.
- Case 2: '(' (Opening Bracket) → Push to stack.
- Case 3: ')' (Closing Bracket) → Pop from stack and push to postfix until '(' is found, then remove '('.
- Case 4: Operator (+, -, *, /, ^):
  - If stack is empty, push operator in stack.
  - Else, pop from stack and push into postfix while stack[top] has higher or equal precedence.
  - Push the current operator after loop end.

Step 4: End of Expression

- After the infix expression ends:
- Pop all remaining elements from the stack.
- Push them into the postfix expression.

Infix Expression:
*(A + B) * (C ^ D - E) ^ (F + G * H) - I*

Step	Infix	Stack	Postfix
1	(	(	
2	A	(	A
3	+	(,+	A
4	B	(,+	AB
5	)		AB+
6	*	*	AB+
7	(	*(,	AB+
8	C	*(,	AB+C
9	^	*(,^	AB+C
10	D	*(,^	AB+CD
11	-	*(,-	AB+CD^
12	E	*(,-	AB+CD^E
13	)	*	AB+CD^E-
14	^	*^	AB+CD^E-
15	(	*^,(,	AB+CD^E-F
16	F	*^,(,	AB+CD^E-F
17	+	*^,(,+	AB+CD^E-F
18	G	*^,(,+	AB+CD^E-FG
19	*	*^,(,+*	AB+CD^E-FG
20	H	*^,(,+*	AB+CD^E-FGH
21	)	*^	AB+CD^E-FGH*+
22	-	-	AB+CD^E-FGH*+^*
23	I	-	AB+CD^E-FGH*+^*I

Coding Part

```
// for all cases study the algorithm and cases written in ppt

#include<iostream>
#include <ctype.h>   //isalphanum
using namespace std;
#define size 100
int stack[size];
int top = -1;

void push(char c){
   top++;
   stack[top] = c;
}

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

```
void infixToPostFix(char infix[], char postfix[]){
   int i=0;
   int j=0;
   while(infix[i]!='\0'){
      char ch = infix[i];

      // Case 1: Operand (letter or digit) → Push to postfix.
      if(isalnum(ch)){  //can check a character is alphabet or numeric and return true or false
         postfix[j] = ch;
         j++;
      }
      // Case 2: '(' (Opening Bracket) → Push to stack.
      else if(ch == '('){
         push(ch);
      }

      // Case 3: ')' (Closing Bracket) → Pop from stack and push into postfix until '(' is found, then remove '('.

      else if(ch == ')'){
         while(stack[top]!='('){
            char popped = stack[top];
            postfix[j] = popped;
            top--;
            j++;
         }
         top--;
      }
```

```
// Case 4: Operator (+, -, *, /, ^):
//      - If stack is empty, push operator in stack.
//    - Else, pop from stack and push to postfix while stack[top] has higher or equal precedence.
//   - Push the current operator after loop end.

      else{
         while(top!=-1 && precedene(stack[top]) >= precedene(ch) ){
            char popped = stack[top];
            postfix[j] = popped;
            j++;
            top--;
         }
         push(ch);
      }
      i++;
   }


   // Case-5 or step4 : After the infix expression ends:Pop all remaining elements from the stack.Push them into the
postfix expression.

   while(top!=-1){
      postfix[j] = stack[top];
      top--;
      j++;
   }
    postfix[j] = '\0'; // '\0' means postfix array of characters end here

}
```

```
int main(){
    char infix[size], postfix[size];
    cout<<"enter the pattern : ";
    cin>>infix;

    infixToPostFix(infix, postfix);

    cout<<endl<<postfix;

}
```