Infix to Postfix

infix exp → ((A+B) -C * (D/E)) +F

exp → ((A+B) -C * (D/E)) +F

(* - /) ( )

algorithm →

1. Scan the exp from left to right.
2. if incoming character is alphabet or digit (operand) then append in the postfix exp.
3. if incoming character is open '(' (brackets) the push into the stack.
4. if incoming charatter is close ')' then pop the stack untill onpen '(' is found. and poped element append to the postfix. then after pop the stack.
5. if incoming character is high perority compare than the top of the stack, then incoming charakter push in the stack.
6. If incoming charactor is less than equal to compare than the top of the stack, pop the stack and append to the postfix exp. until incoming character become higher perority. the push the incoming character into the stack.
7. If there are no incoming character then pop the stack and append to the postfix exp until stack is empty.

```
public class InfixToPostfix{
public int checkPriority(char ch){
      switch(ch){
      case '+':
      case '-':
          return 1;
      case '*'
      case '/':
          return 2;
      case '^':
          return 3;
      }
      return -1;
}

public String infixToPostfixOperation(String exp){
   →String postfixExp="";
   →Stack<Character> stack=new Stack<>();
   for(int i=0; i<str.length; i++){
       char ch = str.charAt(i);
     if(Character.isLetterOrDigit(ch)){
        postfixExp += ch;
     }else if(ch == '('){
        stack.push(ch);
     }else if(ch == ')'){
      while(!stack.isEmpty() && stack.peek() != '('){
         PostfixExp += stack.peek();
         stack.pop();
      }
      stack.pop();
     }else{
        while(!stack.isEmpty() && checkPriority(ch) <=
                    checkPriority(stack.peek())){
          postfix += stack.peek();
          stack.pop();
        }
        stack.push(ch);
   }}
  while(!stack.isEmpty()){
    postfixExp += stack.peek();
    stack.pop();
  }
  return postfixExp;
}
```

A+B-(C+D)+E → infix

A → 65
Z → 90
a → 97
z → 122
0 → 48
9 → 57

```
String exp[] = { "A*B-(C+D)+E", "((A+B)-C*(D/E))+F",
"(A+B)*C-(D-E)*(F+G)", "(((A+B)*C)-((D-E)*(F+G)))",    "A+B*C/D-F+A^E" };
```

Infix expression