Algorithm 2 (Standard Algorithm):

1. Scan the given prefix expression from right to left character by character.
2. If the character is an operand, push it into the stack.
3. But if the character is an operator:
4. Pop the top two values from stack.
5. Concatenate this operator with these two values (1st top value+operator+2nd top value) to get a new string.
6. Now push this resulting string back into the stack.
7. Repeat this process until the end of prefix expression. Now the value in the stack is the desired infix expression.

Algorithm 1 (Has some issues in braces in final answer in some test cases):

1. Create 1 stack: oprtorstk – chr stack for holding operators
2. Create an array of sufficient size to hold infix expression: infix

Declare an integer variable to point to latest index of infix filled and initialize to 0: last\_index = 0

1. Read each character of entered prefix expression one-by-one from start to end.
2. If chr == ‘+’ || ‘-‘ || ‘\*’ || ’/’ || ‘$’ :
3. Push chr in oprtotrstk.
4. Append ‘(‘ to infix.
5. Last\_index ++.
6. Else if chr == operand :
7. Append chr to infix.
8. Last\_index ++.
9. If infix[last\_index-1]==’(‘ : do nothing

Else if infix[last\_index-1]== ’+’ || ‘-‘ || ‘\*’ || ‘/’ || ‘$’: Append ‘)’ to infix.

1. Pop oprtorstk and append to infix.
2. After reading all characters in entered prefix, infix will contain equivalent infix expression.