Lab program 2:

Write a program to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide)

```
#include <stdio.h>
#include <conio.h>
#include <string.h>

int index = 0, pos = 0, top = -1, length;
char symbol, temp, infix[30], postfix[30], stack[30];

void infixToPostfix();
void push(char symbol);
char pop();
int precedence(char symb);

int main() {
   printf("Enter infix expression:\n");
   scanf("%s", infix);

   infixToPostfix();
   printf("\nInfix expression:\n%s", infix);
   printf("\nPostfix expression:\n%s", postfix);
   return 0;
}

void infixToPostfix() {
   length = strlen(infix);
   push('#');
```

```
while (index < length) {
   symbol = infix[index];
   switch (symbol) {
      case '(':
         push(symbol);
         break;

      case ')':
         temp = pop();
         while (temp != '(') {
            postfix[pos++] = temp;
            temp = pop();
         }
         break;

      case '+':
      case '-':
      case '*':
      case '/':
      case '^':
         while (precedence(stack[top]) >= precedence(symbol)) {
            temp = pop();
            postfix[pos++] = temp;
         }
         push(symbol);
         break;
      default:
         postfix[pos++] = symbol;
   }
   index++;
```

```
    }

    while (top > 0) {
        temp = pop();
        postfix[pos++] = temp;
    }
    postfix[pos] = '\0';
}

void push(char symbol) {
    top = top + 1;
    stack[top] = symbol;
}

char pop() {
    char symb = stack[top];
    top--;
    return symb;
}

int precedence(char symbol) {
    int p;
    switch (symbol) {
        case '^':
            p = 3;
            break;

        case '*':
        case '/':
            p = 2;
```

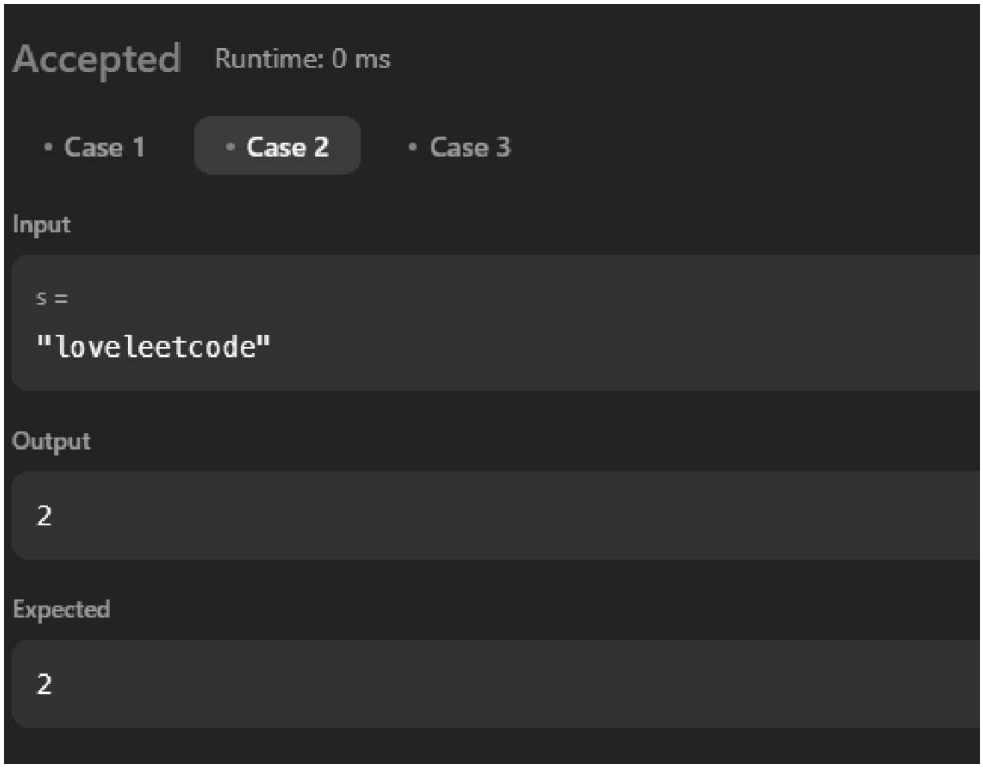
```
        break;

    case '+':
    case '-':
        p = 1;
        break;

    case '(':
        p = 0;
        break;

    case '#':
        p = -1;
        break;
  }
  return p;
}
```

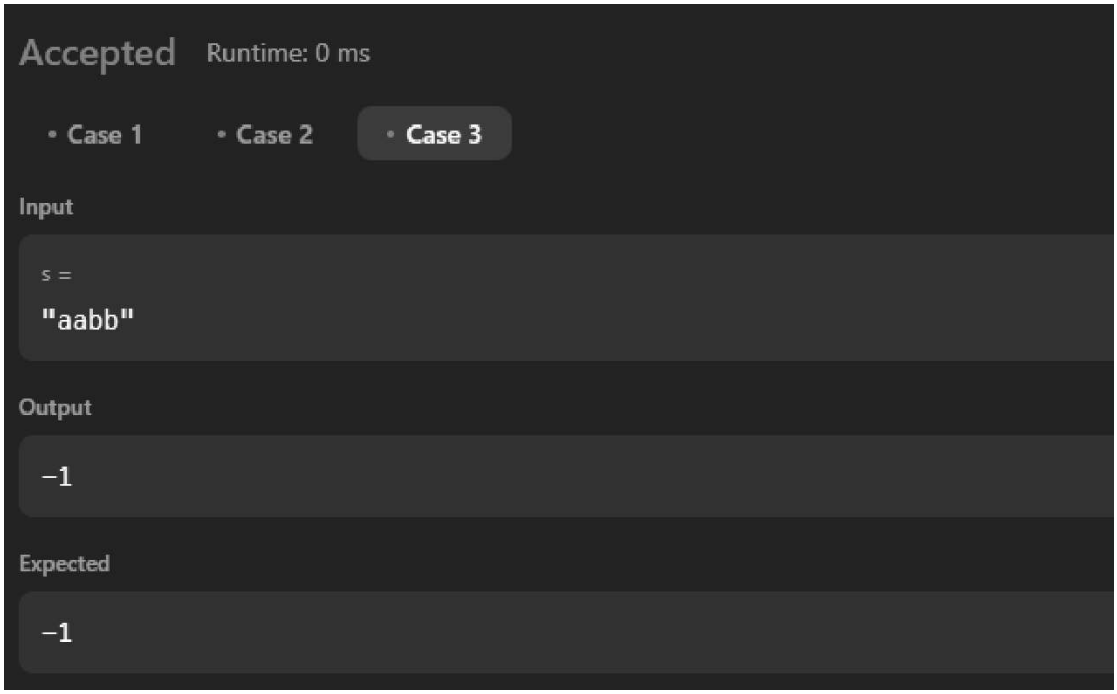
```
Enter infix expression:
a^bc-d+e/f/(g+h)

Infix expression:
a^bc-d+e/f/(g+h)
Postfix expression:
abc^d-ef/gh+/+
Process returned 0 (0x0)   execution time : 73.563 s
Press any key to continue.
```

Leet code :

Given a string s, find the first non-repeating character in it and return its index. If it does not exist, return -1.

```
int firstUniqChar(char* s) {
    int freq[26]={0};
    for(int i=0; s[i]!='\0'; i++){
        freq[s[i]-'a']++;
    }
    for(int i=0; s[i]!='\0'; i++){
        if(freq[s[i]-'a']==1){
            return i;
        }
    }
    return -1;
}
```


Accepted
Runtime: 0 ms
• Case 1
• Case 2
• Case 3
Input
s =
"loveleetcode"
Output
2
Expected
2


Accepted
Runtime: 0 ms
• Case 1
• Case 2
• Case 3
Input
s =
"aabb"
Output
-1
Expected
-1