Rajalakshmi Engineering College

Name: Piraisoodan R
Email: 240701384@rajalakshmi.edu.in
Roll no: 240701384
Phone: 8056892546
Branch: REC
Department: I CSE FD
Batch: 2028
Degree: B.E - CSE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 3_CY

Attempt : 1
Total Mark : 30
Marks Obtained : 30

Section 1 : Coding

1. Problem Statement

Latha is taking a computer science course and has recently learned about infix and postfix expressions. She is fascinated by the idea of converting infix expressions into postfix notation. To practice this concept, she wants to implement a program that can perform the conversion for her.

Help Latha by designing a program that takes an infix expression as input and outputs its equivalent postfix notation.

Example

Input:

(3+4)5

Output:

34+5

Input Format

The input consists of a string, the infix expression to be converted to postfix notation.

Output Format

The output displays a string, the postfix expression equivalent of the input infix expression.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: A+B*C-D/E

Output: ABC*+DE/-

Answer

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
#define max 100
typedef struct {
   int top;
   char items[max];
}Stack;
void initStack(Stack*stack){
   stack->top=-1;
}
int  isempty(Stack* stack){
   return stack->top==-1;
}
void push(Stack*stack,char item){
   if(stack->top<max-1){
      stack->items[++stack->top]=item;
   }
}
char pop(Stack*stack){
    if(!isempty(stack)){
        return stack->items[stack->top--];
    }
    return '\0';
}
char peep(Stack*stack){
    if(!isempty(stack)){
        return stack->items[stack->top];
    }
    return '\0';
}
int precedence(char operators){
    switch(operators){
        case '+':
        case '-':
        return 1;
        case '*':
        case '/':
        return 2;
        case '^':
        return 3;
        default:
        return 0;

    }
}
void infixtopostfix(char*infix, char*postfix){
    Stack stack;
    initStack(&stack);
    int j=0;
    for(int i=0;infix[i];i++){
        char c=infix[i];
        if(isalnum(c)){
            postfix[j++]=c;
        }
        else if(c=='('){
            push(&stack,c);
        }
        else if(c==')'){
            while(!isempty(&stack)&& peep(&stack)!='('){
                postfix[j++]=pop(&stack);
            }
            pop(&stack);
        }
        else{
            while(!isempty(&stack)&& precedence(peep(&stack))>=precedence(c)){
                postfix[j++]=pop(&stack);
            }
            push(&stack,c);
        }
    }
    while(!isempty(&stack)){
        postfix[j++]=pop(&stack);
    }
    postfix[j]='\0';
}
int main(){
    char infix[max];
    char postfix[max];
    printf("");
    fgets(infix,sizeof(infix),stdin);
    infix[strcspn(infix,"\n")]=0;
    infixtopostfix(infix,postfix);
    printf("%s\n",postfix);
    return 0;
}
```

Status : Correct **_Marks : 10/10_**

2. Problem Statement

Suppose you are building a calculator application that allows users to enter mathematical expressions in infix notation. One of the key features of your calculator is the ability to convert the entered expression to postfix notation using a Stack data structure.

Write a function to convert infix notation to postfix notation using a Stack.

Input Format

The input consists of a string, an infix expression that includes only digits(0-9), and operators(+, -, *, /).

Output Format

The output displays the equivalent postfix expression of the given infix expression.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1+2*3/4-5
Output: 123*4/+5-

Answer

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

#define MAX 100

char stack[MAX];
int top = -1;

void push(char c) {
   if (top < MAX - 1) {
      stack[++top] = c;
   }
}

char pop() {
   if (top >= 0) {
      return stack[top--];
   }
   return '\0';
}

char peek() {
   if (top >= 0) {
      return stack[top];
   }
   return '\0';
```

```
}

int precedence(char op) {
   switch(op) {
      case '+':
      case '-': return 1;
      case '*':
      case '/': return 2;
      default : return 0;
   }
}

void infixToPostfix(char* infix) {
   char postfix[MAX];
   int j = 0;
   int len = strlen(infix);

   for (int i = 0; i < len; i++) {
      char c = infix[i];

      if (isdigit(c)) {
         postfix[j++] = c;
      }

      else if (c == '(') {
         push(c);
      }

      else if (c == ')') {
         while (peek() != '(' && top != -1) {
            postfix[j++] = pop();
         }
         pop();
      }

      else if (c == '+' || c == '-' || c == '*' || c == '/') {
         while (precedence(peek()) >= precedence(c)) {
            postfix[j++] = pop();
         }
         push(c);
      }
   }
```

```
    while (top != -1) {
        postfix[j++] = pop();
    }

    postfix[j] = '\0';
    printf("%s\n", postfix);
}


int main() {
    char infix[31];
    scanf("%30s", infix);
    infixToPostfix(infix);
    return 0;
}
```

Status : Correct ***Marks : 10/10***

3. Problem Statement

Buvi is working on a project that requires implementing an array-stack data structure with an additional feature to find the minimum element.

Buvi needs to implement a program that simulates a stack with the following functionalities:

Push: Adds an element onto the stack.Pop: Removes the top element from the stack.Find Minimum: Finds the minimum element in the stack.

Buvi's implementation should efficiently handle these operations with a maximum stack size of 20.

Input Format

The first line of input consists of an integer N, representing the number of elements to push onto the stack.

The second line consists of N space-separated integer values, representing the elements to be pushed onto the stack.

Output Format

The first line of output displays "Minimum element in the stack: " followed by the minimum element in the stack after pushing all elements.

The second line displays "Popped element: " followed by the popped element.

The third line displays "Minimum element in the stack after popping: " followed by the minimum element in the stack after popping one element.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 4
5 2 8 1
Output: Minimum element in the stack: 1
Popped element: 1
Minimum element in the stack after popping: 2

Answer

```
#include <stdio.h>
#include <limits.h>

#define MAX 20

int stack[MAX];
int minStack[MAX];
int top = -1;
int minTop = -1;

void push(int x) {
   if (top < MAX - 1) {
      stack[++top] = x;


      if (minTop == -1 || x <= minStack[minTop]) {
         minStack[++minTop] = x;
      }
   }
```

```
}

int pop() {
    if (top == -1) {
        return -1;
    }
    int popped = stack[top--];


    if (popped == minStack[minTop]) {
        minTop--;
    }

    return popped;
}

int getMin() {
    if (minTop == -1) {
        return -1;
    }
    return minStack[minTop];
}

int main() {
    int N;
    scanf("%d", &N);

    int val;
    for (int i = 0; i < N; i++) {
        scanf("%d", &val);
        push(val);
    }


    printf("Minimum element in the stack: %d\n", getMin());


    int popped = pop();
    printf("Popped element: %d\n", popped);


    printf("Minimum element in the stack after popping: %d\n", getMin());
    return 0;
}
```

Status : Correct ***Marks : 10/10***