Rajalakshmi Engineering College

Name: NITHISH RAJ L
Email: 240701366@rajalakshmi.edu.in
Roll no: 2116240701366
Phone: 8072719523
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 <ctype.h>
#include <string.h>

#define MAX 100

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

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

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 '^': return 3;
        case '*':
        case '/': return 2;
        case '+':
        case '-': return 1;
        default: return 0;
    }

}

int isLeftAssociative(char op)
{
    return op != '^';
}


void infixToPostfix(char* infix, char* postfix)
{
    int i, k = 0;
    char ch;
    for (i = 0; infix[i]; i++)
    {
        ch = infix[i];
        if (isalnum(ch))
```

```
        {
            postfix[k++] = ch;
        }
        else if (ch == '(')
        {
            push(ch);
        }
        else if (ch == ')')
        {
            while (top != -1 && peek() != '(')
            {
                postfix[k++] = pop();
            }
            pop();
        }
        else
        {
            while (top != -1 && precedence(peek()) > 0 && (precedence(peek()) >
precedence(ch) || (precedence(peek()) == precedence(ch) &&
isLeftAssociative(ch))))
            {
                postfix[k++] = pop();
            }
            push(ch);
        }
    }
    while (top != -1)
    {
        postfix[k++] = pop();
    }
    postfix[k] = '\0';
}

int main()
{
    char infix[MAX], postfix[MAX];
    scanf("%s", infix);
    infixToPostfix(infix, postfix);
    printf("%s\n", postfix);
    return 0;
}
```

Status : Correct *Marks : 10/10*

2. 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>
#define MAX 20
int stack[MAX];
int top=-1;

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

int pop()
{
   if(top>=0)
   {
      int popped=stack[top];
      top--;
      return popped;
   }
   return -1;
}

int findMin()
{
   int min=stack[0];
   for(int i=1;i<=top;i++)
   {
      if(stack[i]<min)
      {
         min=stack[i];
      }
   }
   return min;
}
```

```
int main()
{
    int n,i,popped_element;
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        int value;
        scanf("%d",&value);
        push(value);
    }
    printf("Minimum element in the stack: %d\n",findMin());
    popped_element=pop();
    printf("Popped element: %d\n",popped_element);
    if(top>=0)
    {
        printf("Minimum element in the stack after popping: %d\n",findMin());
    }
    else
    {
        printf("Stack is empty after popping.\n");
    }
    return 0;
}
```

Status : Correct *Marks : 10/10*

3. Problem Statement

In an educational setting, Professor Smith tasks Computer Science students with designing an algorithm to evaluate postfix expressions efficiently, fostering problem-solving skills and understanding of stack-based computations.

The program prompts users to input a postfix expression, evaluates it, and displays the result, aiding students in honing their coding abilities.

Input Format

The input consists of the postfix mathematical expression.

The expression will contain real numbers and mathematical operators (+, -, *, /), without any space.

Output Format

The output prints the result of evaluating the given postfix expression.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 82/
Output: 4

Answer

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

#define MAX 100
int stack[MAX];
int top=-1;

void push(int value)
{
```

```
    if(top==MAX-1)
    {
        printf("Stack Overflow\n");
        exit(1);
    }
    stack[++top]=value;
}

int pop()
{
    if(top==-1)
    {
        printf("Stack Underflow\n");
        exit(1);
    }
    return stack[top--];
}

int evaluatePostfix(char* expr)
{
    for(int i=0;expr[i]!='\0';i++)
    {
        char ch=expr[i];
        if(isdigit(ch))
        {
            push(ch-'0');
        }

        else
        {
            int val2=pop();
            int val1=pop();
            switch(ch)
            {
                case '+':
                push(val1+val2);
                break;

                case '-':
                push(val1-val2);
                break;
```

```
            case '*':
            push(val1*val2);
            break;

            case '/':
            push(val1/val2);
            break;

            default:
            printf("Invalid character in expression\n");
            exit(1);
        }
      }
   }
   return pop();
}

int main()
{
   char expr[100];
   scanf("%s",expr);
   int result=evaluatePostfix(expr);
   if(result==(int)result)
   {
      printf("%d\n",(int)result);
   }
}
```

Status : Correct *Marks : 10/10*