Rajalakshmi Engineering College

Name: Sankara Gomathi R
Email: 240701470@rajalakshmi.edu.in
Roll no: 240701470
Phone: 7530026101
Branch: REC
Department: I CSE FE
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 : 20

Section 1 : Coding

1. 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>
typedef struct node{
   char data;
   struct node*next;
}node;
node*top=0;
void push(char ch){
   node*newnode=(node*)malloc(sizeof(node));
   newnode->data=ch;
   newnode->next=top;
   top=newnode;
}

char pop(){
   if(top==0)return '\0';
   char popped=top->data;
   node*temp=top;
   top=top->next;
   free(temp);
   return popped;
}

char peek(){
   if(top==0)return '\0';
   return top->data;
}
```

```
int isempty(){
    return top==0;
}

int operators(char ch){
    return ch=='+'||ch=='-'||ch=='*'||ch=='/';
}

int precedence(char op){
    switch(op){
        case'^':
        return 3;
        case'*':
        case'/':
        return 2;
        case'+':
        case'-':
        return 1;
        default:
        return 0;
    }
}

void infixpostdix(char*infix){
    int i=0,j=0;
    char ch;
    char postfix[100];
    while((ch=infix[i++])!='\0'){
        if(isalnum(ch)){
            postfix[j++]=ch;
        }
        else if (ch=='('){
            push(ch);
        }
        else if(ch==')'){
            while(!isempty() && peek()!='('){
                postfix[j++]=pop();
            }
            pop();
        }
        else if(operators(ch)){
            while(!isempty() && precedence(peek())>=precedence(ch)){
                postfix[j++]=pop();
            }
            push(ch);
        }
    }
    while(!isempty()){
        postfix[j++]=pop();
    }
    postfix[j]='\0';
    printf("%s\n",postfix);
}

int main(){
    char infix[100];
    scanf("%s",infix);
    infixpostdix(infix);
    return 0;
}
```

Status : Wrong **_Marks : 0/10_**

2. Problem Statement

Rithi is building a simple text editor that allows users to type characters, undo their typing, and view the current text. She has implemented this text editor using an array-based stack data structure.

She has to develop a basic text editor with the following features:

Type a Character (Push): Users can type a character and add it to the text editor.Undo Typing (Pop): Users can undo their typing by removing the last character they entered from the editor.View Current Text (Display): Users can view the current text in the editor, which is the sequence of characters in the buffer.Exit: Users can exit the text editor application.

Write a program that simulates this text editor's undo feature using a character stack and implements the push, pop and display operations accordingly.

Input Format

The input consists of integers corresponding to the operation that needs to be performed:

Choice 1: Push the character onto the stack. If the choice is 1, the following input is a space-separated character, representing the character to be pushed onto the stack.

Choice 2: Pop the character from the stack.

Choice 3: Display the characters in the stack.

Choice 4: Exit the program.

Output Format

The output displays messages according to the choice and the status of the stack:

1. If the choice is 1, print: "Typed character: <character>" where <character> is the character that was pushed to the stack.
2. If the choice is 2, print: "Undo: Removed character <character>" where <character> is the character that was removed from the stack.
3. If the choice is 2, and if the stack is empty without any characters, print "Text editor buffer is empty. Nothing to undo."
4. If the choice is 3, print: "Current text: <character1> <character2> ... <characterN>" where <character1>, <character2>, ... are the characters in the stack, starting from the last pushed character.
5. If the choice is 3, and there are no characters in the stack, print "Text editor buffer is empty."
6. If the choice is 4, exit the program.
7. If any other choice is entered, print "Invalid choice"

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1 H
1 A
3
4
Output: Typed character: H

```
Typed character: A
Current text: A H
```

Answer

```
#include<stdio.h>
#include<stdlib.h>
typedef struct node{
   char data;
   struct node*next;
}node;
node*top=0;
void push(char val){
   node*newnode=(node*)malloc(sizeof(node));
   newnode->data=val;
   newnode->next=top;
   top=newnode;
   printf("Typed character: %c\n",val);
}

void pop(){
   if(top==0){
      printf("Text editor buffer is empty.Nothing to undo.\n");
      return ;
   }
   node*temp=top;
   top=top->next;
   printf("Undo:Removed character %c\n",temp->data);
   free(temp);
}

void display(){
   if(top==0){
      printf("Text editor buffer is empty.\n");
      return ;
   }
   node*temp=top;
   printf("Current text:");
   while(temp!=0){
      printf("%c",temp->data);
      temp=temp->next;
   }
   printf("\n");
}

int main(){
   int choice;
   char val;
   while(1){
      scanf("%d",&choice);
      getchar();
      switch(choice){
         case 1:
         scanf("%c",&val);
         push(val);
         break;
         case 2:
         pop();
         break;
         case 3:
         display();
         break;
         case 4:
         return 0;
         default:
         printf("Invalid choice\n");
      }
   }
}
```

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_SIZE 100
int stack[MAX_SIZE];
int minstack[MAX_SIZE];
int top=-1;
int mintop=-1;
void push(int value){
   if(top>=MAX_SIZE-1){
      printf("Stack overflow\n");
```

```
        return;
    }
    stack[++top]=value;
    if(mintop==-1||value<=minstack[mintop]){
        minstack[++mintop]=value;
    }
}

int pop(){
    if(top==-1){
        printf("Stack underflow\n");
        return INT_MIN;
    }
    int value=stack[top--];
    if(value==minstack[mintop]){
        mintop--;
    }
    return value;
}

int get_min(){
    if(mintop==-1){
        printf("Stack is empty\n");
        return INT_MIN;
    }
    return minstack[mintop];
}

void printstac(){
    printf("Current stack:[");
    for(int i=0;i<=top;i++){
        printf("%d",stack[i]);
        if(i<top)printf(",");
    }
    printf("]\n");
}

int main(){
    int n,value;
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        scanf("%d",&value);
        push(value);
    }
    printf("Minimum element in the stack: %d\n",get_min());
    int popped=pop();
    printf("Popped element: %d\n",popped);
    printf(" Minimum element in the stack after popping: %d\n",get_min());
    return 0;
}
```

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