Rajalakshmi Engineering College

Name: NEIL DANIEL A
Email: 240701356@rajalakshmi.edu.in
Roll no: 240701356
Phone: 8925059757
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

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 <stdlib.h>

#define MAX_SIZE 20

typedef struct {
    int data[MAX_SIZE];
    int minData[MAX_SIZE];
    int top;
} Stack;

void initStack(Stack* stack) {
    stack->top = -1;
}
```

```
int isFull(Stack* stack) {
    return stack->top == MAX_SIZE - 1;
}

int isEmpty(Stack* stack) {
    return stack->top == -1;
}

void push(Stack* stack, int value) {
    if (isFull(stack)) {
        printf("Stack is full!\n");
        return;
    }
    stack->data[++stack->top] = value;
    if (stack->top == 0) {
        stack->minData[stack->top] = value;
    } else {
        stack->minData[stack->top] = value < stack->minData[stack->top - 1] ?
value : stack->minData[stack->top - 1];
    }
}

int pop(Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty!\n");
        return -1;
    }
    return stack->data[stack->top--];
}

int findMinimum(Stack* stack) {
    if (isEmpty(stack)) {
        return -1;
    }
    return stack->minData[stack->top];
}

int main() {
    Stack stack;
    initStack(&stack);

    int N;
```

```
    scanf("%d", &N);

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

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

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

    printf("Minimum element in the stack after popping: %d\n",
findMinimum(&stack));

    return 0;
}
```

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

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

Raj is a software developer, and his team is building an application that processes user inputs in the form of strings containing brackets. One of the essential features of the application is to validate whether the input string meets specific criteria.

During testing, Raj inputs the string "(([])){}". The application correctly returns "Valid string" because the input satisfies the criteria: every opening bracket (, [, and { has a corresponding closing bracket),], and }, arranged in the correct order.

Next, Raj tests the application with the string "([)]". This time, the

application correctly returns "Invalid string" because the opening bracket [is incorrectly closed by the bracket), which violates the validation rules.

Finally, Raj enters the string "{[()]}". The application correctly identifies it as a "Valid string" since all opening brackets are matched with the corresponding closing brackets in the correct order.

As a software developer, Raj's responsibility is to ensure that the application works reliably and produces accurate results for all input strings, following the validation rules. He accomplishes this by using a method for solving such problems.

Input Format

The input comprises a string representing a sequence of brackets that need to be validated.

Output Format

The output prints "Valid string" if the string is valid. Otherwise, it prints "Invalid string".

Refer to the sample output for formatting specifications.

Sample Test Case

Input: (([])){}
Output: Valid string

Answer

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
int stack[100],top=-1;
void push(int ch){
stack[++top]=ch;
}
int pop(){
  return stack[top--];
```

```
}
int isempty(){
  return top==-1;
}
int main(){
  int flag=0;
  char expr[100];
scanf("%s",expr);
  for(int i=0;i<strlen(expr);i++){
   if(expr[i]=='(' || expr[i]=='[' || expr[i]=='{'){
   push(expr[i]);
     }
   else if(expr[i]==')' || expr[i]==']' || expr[i]=='}'){
   char c=pop();
   if((c==-1) || (c=='(' && expr[i]!=')') || (c=='[' && expr[i]!=']') || (c=='{' && expr[i]!='}')){
   flag=1;
   break;
      }
   }
}
  if(!isempty()){
     flag=1;
  }
if(flag){
printf("Invalid string");
 }
 else{
printf("Valid string");
 }
}
```

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