GGU

2.2.6. Reversing a string using a stack. 01:33

Complete the missing code of `void reverse(char [] str)` function which reverses a string using the `push()` and `pop()` functions.

Sample Test Cases +

Question Hints +

Explorer

ReverseStri... Submit

Debugger

```
#include<stdio.h>
#include<string.h>
#define STACK_MAX_SIZE 30
char arr[STACK_MAX_SIZE];
int top = -1;
void push(char element) {
    if(top == STACK_MAX_SIZE - 1) {
        printf("Stack is overflow.\n");
    } else {
        top = top + 1;
        arr[top] = element;
    }
}
char pop() {
    long int x;
    if(top < 0) {
        printf("Stack is underflow.\n");
        return -1;
    } else {
        x = arr[top];
        top = top - 1;
        return x;
    }
}
void reverse(char str[]) {
    int i;
    for(i=0;str[i]!='\0';i++){
        push(str[i]);
    }
    for(i=0;str[i]!='\0';i++){
        str[i]=pop();
    }
}
void main() {
    char ch[80], temp;
    printf("Enter a string : ");
    scanf("%s", ch);
    reverse(ch);
    printf("The reverse of a given string : %s\n", ch);
}
```

Prev Reset Submit Next

GGU

2.2.7. Check for the balanced parenthesis usi... 20:09

Complete the code in `int isBalanced(char exp[])` function which checks for the balanced parenthesis using the `push()`, `pop()` and `isempty()` functions.

Sample Test Cases +

Question Hints +

Explorer

BalancedP...

Submit

Debugger

```
#include<stdio.h>
#include<string.h>
#define STACK_MAX_SIZE 30
char arr[STACK_MAX_SIZE];
int top = -1;

void push(char element) {
    if(top == STACK_MAX_SIZE - 1) {
        printf("Stack is overflow.\n");
    } else {
        top = top + 1;
        arr[top] = element;
    }
}

char pop() {
    long int x;
    if(top < 0) {
        printf("Stack is underflow.\n");
        return -1;
    } else {
        x = arr[top];
        top = top - 1;
        return x;
    }
}

int isempty() {
    if(top == -1)
        return 1;
    else
        return 0;
}

//Complete the code below.
int isBalanced(char exp[]) {
    char ch;
    for(int i=0;exp[i]!='\0';i++){
        ch=exp[i];
        if(ch=='('||ch=='{'||ch=='[')
        {
            push(ch);
        }
        else if(ch==')'||ch=='}'||ch==']'){
            if(isempty())
                return 0;
            char topChar=pop();
            if((ch==')'&& topChar!='('||
                (ch=='}'&& topChar!='{'||
                (ch==']'&& topChar!='['){
                return 0;
            }
        }
    }
    return isempty();
}

void main() {
    char ch[80], temp;
    printf("Enter an expression : ");
    scanf("%s", ch);
    if(isBalanced(ch) == 1) {
        printf("Expression is balanced.\n");
    } else {
        printf("Expression is not balanced.\n");
    }
}
```

Prev Reset Submit Next

GGU

2.3.2. Implementation of Queue using Linked... 05:50

Fill the missing code in the functions `dequeue()`, `size()` and `isEmpty()` in the below code.

The function `dequeue()` removes an element from the queue and it gives **"Queue is underflow."** error if there are no elements in the queue.

The function `size()` prints the current size of the queue i.e. is the number of elements in the queue.

The function `isEmpty()` prints whether the queue is empty or not.

Sample Test Cases +

Question Hints +

Explorer

QueueList... QueueList... Submit

```
typedef struct queue *Q;
Q front = NULL, rear = NULL;

void enqueue(int element) {
    Q temp = NULL;
    temp = (Q)malloc(sizeof(struct queue));
    if(temp == NULL) {
        printf("Queue is overflow.\n");
    } else {
        temp -> data = element;
        temp -> next = NULL;

        if(front == NULL) {
            front = temp;
        } else {
            rear -> next = temp;
        }
        rear = temp;
        printf("Successfully inserted.\n");
    }
}

void dequeue() {
    Q temp;
    if(front==NULL){
        printf("Queue is underflow.\n");
    }
    else{
        temp=front;
        printf("Deleted value = %d\n",temp->data);
        front=front->next;
        free(temp);
        if(front==NULL){
            rear=NULL;
        }
    }
}
void display() {
    if(front == NULL) {
        printf("Queue is empty.\n");
    } else {
        Q temp = front;
        printf("Elements in the queue : ");
        while(temp != NULL) {
            printf("%d ", temp -> data );
            temp = temp -> next;
        }
        printf("\n");
    }
}
void size() {
    Q temp=front;
    int count=0;
    while(temp!=NULL){
        count++;
        temp=temp->next;
    }
    printf("Queue size : %d\n",count);
}

void isEmpty() {
    if(front==NULL){
        printf("Queue is empty.\n");
    }
    else{
        printf("Queue is not empty.\n");
    }
}

```

Debugger Plots

Prev Reset Submit Next

GGU

2.3.3. C program to implement different Oper... 08:03

Write a program to implement a queue using **linked lists**.
Implement the queue operations as separate functions:

- `enqueue(int element)`: Adds an element to the queue.
- `dequeue()`: Removes and displays the element at the front of the queue.
- `display()`: Prints all the elements in the queue.
- `isEmpty()`: Displays whether the queue is empty or not.
- `size()`: Displays the number of elements currently in the queue.

Note:
The partial code has been provided to you in the editor, you are required to fill in the missing code in the function bodies.

Sample Test Cases +

Explorer

QueueUsin... QueueOper... Submit

Debugger

```c
struct queue {
    int data;
    struct queue *next;
};

typedef struct queue *Q;
Q front = NULL, rear = NULL;

void enqueue(int element) {
Q newNode =(Q)malloc(sizeof(struct queue));
    newNode->data=element;
    newNode->next=NULL;
    if(rear==NULL){
        front=rear=newNode;
    }
else{
    rear->next=newNode;
    rear=newNode;
}
    printf("Successfully inserted.\n");
}

void dequeue() {
if(front==NULL){
    printf("Queue is underflow.\n");
    return;
}
    Q temp=front;
    printf("Deleted value = %d\n",front->data);
    front=front->next;
    free(temp);
    if(front==NULL){
        rear=NULL;
    }

}

void size() {
int count=0;
    Q temp=front;
    while(temp!=NULL){
        count++;
        temp=temp->next;
    }
    printf("Queue size : %d\n",count);

}

void isEmpty() {
if(front==NULL)
    printf("Queue is empty.\n");
    else
    printf("Queue is not empty.\n");

}

void display() {
    if(front == NULL) {
        printf("Queue is empty.\n");
    } else {
        Q temp = front;
        printf("Elements in the queue : ");
        while(temp != NULL) {
            printf("%d ", temp -> data);
            temp = temp -> next;
        }
        printf("\n");
    }
}
```

Prev Reset Submit Next

GGU

2.3.4. Implementation of Queues using two s... 18:39

A queue can be implemented using two stacks.

Enqueue operation is defined as follows:

1. Push the element to first stack.

Dequeue operation is defined as follows:

1. If the second stack is not empty, then pop the element from the second stack.
2. If the second stack is empty , then pop all the elements from the first stack and push them into the second stack until first stack is empty.
3. Finally pop the element from the second stack.

Display operation is defined as follows:

1. First print all the elements from the first stack.
2. Then print all the elements from the second stack.

The node of a stack is defined as follows:

```
struct stack {
    int data;
    struct stack *next;
};
//Creating an alias for the pointer to the struct stack
typedef struct stack *stk;
```

The queue which is to be implemented using two stacks is defined as follows:

```
struct queue{
    stk stack1;
    stk stack2;
};
//Creating an alias for the pointer to the struct queue
typedef struct queue* Q;
```

Declare a queue variable as follows:

```
Q que;
```

The push operation of the stack is defined as follows:

```
//This function pushes an element x into the stack. T
void push(stk* stack, int x) {
    stk temp; //Declare a variable of type stk.
    temp = (stk)malloc(sizeof(struct stack)); //Allocat
    if(temp == NULL) {
        printf("Stack is overflow.\n"); //Unable to all
    } else {
        temp -> data = x;
        temp -> next = *stack;
        *stack = temp;
    }
}
```

The pop operation of the stack is defined as follows:

```
int pop(stk* stack) {
    stk temp; //Declare a variable of type stk.
    if(stack == NULL) {
        printf("Stack is underflow.\n"); //stack is equ
    } else {
        temp = *stack;
        *stack = temp -> next;
        return temp->data;
    }
}
```

The pseudocode for the enqueue operations using two stack is as follows:

```
que - It is variable of type Q, which has two members v
ele - The element to be enqueued.
void enqueue(int ele) {
    call : push(&(que->stack1),ele); //Simply push the
    Print "Successfully inserted.\n".
}
```

The pseudocode for the dequeue operations using two stack is as follows:

```
que - It is variable of type Q, which has two members v
void dequeue() {
    Declare an integer variable x.
    if(que->stack1 is equal to NULL and que->stack2 is
```

Sample Test Cases +

Explorer

QueueOper... QueueOper... Submit

Debugger

```
1  struct stack {
2      int data;
3      struct stack *next;
4  };
5  typedef struct stack *stk;
6  struct queue{
7      stk stack1;
8      stk stack2;
9  };
10 typedef struct queue* Q;
11 Q que;
12 void push(stk* stack, int x) {
13     stk temp;
14     temp = (stk)malloc(sizeof(struct
       stack));
15     if(temp == NULL) {
16         printf("Stack is
       overflow.\n");
17     }
18     else {
19         temp -> data = x;
20         temp -> next = *stack;
21         *stack = temp;
22     }
23 }
24 int pop(stk* stack) {
25     stk temp;
26     if(*stack == NULL) {
27         printf("Stack is
       underflow.\n");
28         return -1;
29     } else {
30         temp = *stack;
31         *stack = temp -> next;
32         int data=temp->data;
33         free(temp);
34         return data;
35     }
36 }
37 void enqueue(int ele) {
38     push(&(que->stack1),ele);
39     printf("Successfully
       inserted.\n");
40 }
41 void dequeue() {
42     int x;
43     if(que->stack1==NULL&&que-
       >stack2==NULL){
44         printf("Queue is
       underflow.\n");
45         return;
46     }
47     if(que->stack2==NULL){
48         while(que->stack1!=NULL){
49             x=pop(&(que->stack1));
50             push(&(que->stack2),x);}}
51     x=pop(&(que->stack2));
52     printf("Deleted value = %d\n",x);
53     }
54
55
56 void display() {
57     stk temp;
58     if(que->stack1==NULL&&que-
       >stack2==NULL){
59         printf("Queue is empty.\n");
60         return ;
61     }
62     temp=que->stack1;
63     while(temp!=NULL){
64         printf("%d ",temp->data);
65         temp=temp->next;
66     }
67     temp=que->stack2;
68     while(temp!=NULL){
69         printf("%d ",temp->data);
70         temp=temp->next;
71     }
```

Prev Reset Submit Next

The declaration of a node in **CLL** is exactly same as **SLL**

```
struct node {
    int data;
    struct node *next;
 };
typedef struct node *NODE; // typedef defines a new typ
// Instead of struct node * we can use NODE because of
```

As seen in **SLL**, the nodes are dynamically allocated using **malloc()** function in **C** language and it then linked to a chain of nodes which are currently presented in the list.

Now see the `createNodeInCLL()` function, which will create dynamic memory to each node.

Note: Closely observe the `createNodeInCLL()` function and the purpose of each and every line is mentioned in the comments.

```
NODE createNodeInCLL() {
    NODE temp; // Declare a node
    temp = (NODE) malloc(sizeof(struct node)); // Alloc
    temp -> next = NULL; // Make next point to NULL
    return temp; // return the new node
}
```

Closely observe the below program, where we are trying to create a circular linked list with two elements using which we are demonstrating how we are able to traverse to the first node of the list from the last node.

Fill in the missing code in the below function `createNodeInCLL()` in the file `CreateANodeInCLL.c`.

Sample Test Cases +

Explorer

NodeCreati... CreateANo... Submit

Debugger

```c
struct node {
    int data;
    struct node * next;
};
typedef struct node * NODE;

NODE createNodeInCLL() {
NODE temp;
    temp=(NODE)malloc(sizeof(struct node));
    if(temp==NULL){
        printf("memory allocation failed\n");
        exit(1);
    }
    temp->next=temp;
    return temp;
}
void displayCLL(NODE head){
    if(head==NULL){
        printf("List is empty\n");
    return;

}
    NODE current =head;
    printf("The list is : ");
    do{
        printf("%d -> ",current->data);
        current=current->next;
    }
        while(current!=head);
    printf("%d\n",head->data);}
```

2.4.3. Write Code for addNodesInCLL() and tr... 14:05

In the below `circular linked list` program we have two files, one file contains the **main program** and the other file contains the **functions**, to be implemented by the **user**.

Here the user has to implement the code for two functions `addNodesInCLL()` and `traverseListInCLL()`.

The `addNodesInCLL()` function creates a new list and adds elements to the list until delimiter `-1` is occurred.

The `traverseListInCLL()` function traverses and prints all the elements of the list.

Fill in the missing code in the below functions `addNodesInCLL(NODE first, int x)` and `traverseListInCLL(NODE first)` in the file `CreateAndAddNodesInCLL.c`.

Sample Test Cases +

Question Hints +

CircularLL... CreateAnd... Submit

Explorer Debugger

```c
struct node {
    int data;
    struct node *next;
};
typedef struct node *NODE;

NODE createNodeInCLL() {
    NODE temp;
    temp = (NODE)
malloc(sizeof(struct node));
    temp -> next = NULL;
    return temp;
}

NODE addNodesInCLL(NODE first, int
x) {
    NODE newNode =createNodeInCLL();
    newNode->data=x;

    if(first==NULL){
        newNode->next=newNode;
        first=newNode;
    }
    else{
        NODE temp=first;
        while(temp->next!=first){
            temp=temp->next;
        }
        temp->next=newNode;
        newNode->next=first;
    }
    return first;

}


void traverseListInCLL(NODE first) {

    if(first==NULL){
        return;

    }
    NODE temp=first;
    do{
        printf("%d --> ",temp->data);
        temp=temp->next;


    }while(temp!=first);
    printf("\n");


}
```

Prev Reset Submit Next

GGU

2.4.4. Write Code for insertAtBeginInCLL() an... 05:15

Fill in the missing code in the below functions `insertAtBeginInCLL(NODE first, int x)` and `countInCLL(NODE first)` in the file `InsAtBeginAndCountInCLL.c`.

The `insertAtBeginInCLL(NODE first, int x)` function inserts a new node at the beginning of the circular linked list.

The `countInCLL(NODE first)` function counts the number of nodes linked in a circular linked list.

Sample Test Cases +

Question Hints +

Explorer

CircularLL... InsAtBegin... Submit

Debugger Plots

```c
struct node {
    int data;
    struct node *next;
};
typedef struct node *NODE;

NODE createNodeInCLL() {
    NODE temp;
    temp = (NODE) malloc(sizeof(struct node));
    temp -> next = NULL;
    return temp;
}

NODE insertAtBeginInCLL(NODE first, int x) {
    NODE temp= (NODE)malloc(sizeof(struct node));
    temp->data=x;
    if(first==NULL){
        temp->next=temp;
        return temp;
    }
    NODE current=first;
    while(current->next!=first){
        current=current->next;
    }
    current->next=temp;
    temp->next=first;
    return temp;
}

int countInCLL(NODE first) {
    if(first==NULL)
        return 0;
    int sum=1;
    NODE temp=first->next;
    while(temp!=first){
        sum++;
        temp=temp->next;
    }
    return sum;
}

void traverseListInCLL(NODE first) {
    NODE temp = first;
    do {
        printf("%d --> ", temp -> data);
        temp = temp -> next;
    } while (temp != first);
    printf("\n");
}
```

Prev Reset Submit Next

GGU

2.4.5. Write Code for insertAtEndInCLL() fun... 03:26

Fill in the missing code in the below finction `insertAtEndInCLL(NODE first, int x)` in the file `InsAtEndingInCLL.c`, which inserts a new node at the end of circular linked list.

Sample Test Cases +

Question Hints +

Explorer

CircularLL... InsAtEndin... Submit

Debugger

```c
struct node {
    int data;
    struct node *next;
};
typedef struct node *NODE;

NODE createNodeInCLL() {
    NODE temp;
    temp = (NODE)
malloc(sizeof(struct node));
    temp -> next = NULL;
    return temp;
}

NODE insertAtEndInCLL(NODE first,
int x) {
    NODE temp=
(NODE)malloc(sizeof(struct node));
    temp->data=x;
    temp->next=NULL;
    if(first==NULL){
        temp->next=temp;
        return temp;
    }
    NODE current=first;
    while(current->next!=first){
        current=current->next;
    }
    current->next=temp;
    temp->next=first;
    return first;
    // Write your code here

}

void traverseListInCLL(NODE first) {
    NODE temp = first;
    do {
        printf("%d --> ", temp ->
data);
        temp = temp -> next;
    } while (temp != first);
    printf("\n");
}
```

Prev Reset Submit Next

GGU

2.4.8. Write a Program to Create a Double Lin... 01:10

The declaration of a node in DLL is:

```
struct node {
    int data;
    struct node *prev;
    struct node *next;
};
typedef struct node *NODE;
//typedef defines a new type NODE as pointer of struct
//Instead of struct node * we can use NODE because of t
```

Like **singly linked list** and **circular linked list**, each new node of the **double linked list** is dynamically created by using **malloc()** function in **C** language and it then linked to a chain of nodes which are currently presented in the list.

See the `createNodeInDLL()` function of **DLL**, which will create dynamic memory to each node.

Note: Closely observe the `createNodeInDLL()` function also the purpose of each and every line is mentioned in the comments.

```
NODE createNodeInDLL() {
    NODE temp; // declare a node
    temp = (NODE) malloc(sizeof(struct node)); // alloc
    temp -> prev = NULL; // make prev point to NULL
    temp -> next = NULL; // make next point to NULL
    return temp; // return the new node
}
```

Fill in the missing code in the below function `createNodeInDLL()`.

Sample Test Cases +

Explorer

NodeCreati... CreateANo... Submit

Debugger

```
1  struct node {
2      int data;
3      struct node *prev;
4      struct node *next;
5  };
6  typedef struct node * NODE;
7
8  NODE createNodeInDLL() {
9      NODE temp;
10     temp=(NODE)malloc(sizeof(struct
   node));
11     temp->prev=NULL;
12     return temp;
13
14 }
```

Prev Reset Submit Next

2.4.10. Write a Program to Insert an element ... 04:17

Fill in the missing code in the `insertAtBeginInDLL()` and `countInDLL()` methods.

The `insertAtBeginInDLL(NODE first, int x)` function inserts a new integer at the beginning of the double linked list.

The `countInDLL(NODE first)` function counts the number of nodes linked in a double linked list.

Sample Test Cases +

Question Hints +

DoubleLL2.c | InsAtBegin... | Submit

```c
struct node {
    int data;
    struct node *prev;
    struct node *next;
};
typedef struct node * NODE;

NODE createNodeInDLL() {
    NODE temp;
    temp = (NODE)malloc(sizeof(struct node));
    temp -> prev = NULL;
    temp -> next = NULL;
    return temp;
}

int countInDLL(NODE first) {
    int count=0;
    NODE current=first;
    while(current!=NULL){
        count++;
        current=current->next;
    }
    return count;
}

NODE insertAtBeginInDLL(NODE first, int x) {
    NODE temp=(NODE)malloc(sizeof(struct node));
    if(temp==NULL){
        printf("Memory allocation failed.\n");
        return first;
    }
    temp->data=x;
    temp->prev=NULL;
    temp->next=first;
    if(first!=NULL){
        first->prev=temp;
    }
    first=temp;
    return first;
}

void traverseListInDLL(NODE first) {
    NODE lastNode = first;
    while (lastNode != NULL) {
        printf("%d <--> ", lastNode -> data);
        lastNode = lastNode -> next;
    }
    printf("NULL\n");
}
```

Prev | Reset | Submit | Next

GGU

2.4.11. Write a Program to Insert an element ... 11:55

Fill in the missing code in `insertAtPositionInDLL(NODE first, int pos, int x)` function which inserts a new integer at a particular position in the double linked list.

Sample Test Cases +

Question Hints +

Explorer

DoubleLL4.c | InsAtPositi... | Submit

Debugger

```c
struct node {
    int data;
    struct node *prev;
    struct node *next;
};
typedef struct node * NODE;

NODE createNodeInDLL() {
    NODE temp;
    temp = (NODE)malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->next = NULL;
    return temp;
}

NODE insertAtPositionInDLL(NODE first, int position, int x) {
    NODE temp,current;
    int i;
    if(position<1){
        printf("Invalid position.\n");
        return first;
    }
    temp=(NODE)malloc(sizeof(struct node));
    if(temp==NULL){
        printf("Memory allocation failed.\n");
        return first;
    }
    temp->data=x;
    temp->prev=NULL;
    temp->next=NULL;
    if(position==1){
        temp->next=first;
        if(first!=NULL)
            first->prev=temp;
        first=temp;
        return first;
    }
    current=first;
    for(i=1;i<position-1&&current!=NULL;i++){
        current=current->next;
    }
    if(current==NULL){
        printf("No such position in DLL so insertion is not possible\n");
        free(temp);
        return first;
    }
    temp->next=current->next;
    temp->prev=current;
    if(current->next!=NULL)
        current->next->prev=temp;
    current->next=temp;
    return first;
}

void traverseListInDLL(NODE first) {
    NODE lastNode = first;
    while (lastNode != NULL) {
        printf("%d <--> ", lastNode->data);
        lastNode = lastNode -> next;
    }
    printf("NULL\n");
}
```

< Prev | Reset | Submit | Next >