



GGU
GUJARATI GATEWAY
UNIVERSITY

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

ReverseStri...

```
1 #include<stdio.h>
2 #include<string.h>
3 #define STACK_MAX_SIZE 30
4 char arr[STACK_MAX_SIZE];
5 int top = -1;
6 void push(char element) {
7     if(top == STACK_MAX_SIZE - 1) {
8         printf("Stack is
9         overflow.\n");
10    } else {
11        top = top + 1;
12        arr[top] = element;
13    }
14 }
15 char pop() {
16     long int x;
17     if(top < 0) {
18         printf("Stack is
19         underflow.\n");
20         return -1;
21     } else {
22         x = arr[top];
23         top = top - 1;
24         return x;
25     }
26 }
27 void reverse(char str[]) {
28     int i;
29     for(i=0;str[i]!='\0';i++){
30         push(str[i]);
31     }
32     for(i=0;str[i]!='\0';i++){
33         str[i]=pop();
34     }
35 }
36 void main() {
37     char ch[80], temp;
38     printf("Enter a string : ");
39     scanf("%s", ch);
40     reverse(ch);
41     printf("The reverse of a given
42     string : %s\n", ch);
43 }
```

< Prev

Reset

Submit

Next >



GGU
GURU GURU UNIVERSITY

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

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

Explorer

BalancedP...

Submit

```

1  #include<stdio.h>
2  #include<string.h>
3  #define STACK_MAX_SIZE 30
4  char arr[STACK_MAX_SIZE];
5  int top = -1;
6
7  void push(char element){
8      if(top == STACK_MAX_SIZE - 1){
9          printf("Stack is
overflow.\n");
10     } else {
11         top = top + 1;
12         arr[top] = element;
13     }
14 }
15
16 char pop(){
17     long int x;
18     if(top < 0){
19         printf("Stack is
underflow.\n");
20         return -1;
21     } else {
22         x = arr[top];
23         top = top - 1;
24         return x;
25     }
26 }
27
28 int isempty(){
29     if(top == -1)
30         return 1;
31     else
32         return 0;
33 }
34
35 //Complete the code below.
36 int isBalanced(char exp[]){
37     char ch;
38     for(int i=0;exp[i]!='\0';i++){
39         ch=exp[i];
40         if(ch=='(' || ch=='{' || ch=='[')
41         {
42             push(ch);
43         }
44         else
45         if(ch==')' || ch=='}' || ch==']'){
46             if(isempty())
47                 return 0;
48             char topChar=pop();
49             if((ch=='{'&&
topChar!='(')||
(ch=='}'&&
topChar!='}')||
(ch=='['&&
topChar!='[')){
50                 return 0;
51             }
52         }
53     }
54     return isempty();
55 }
56
57 void main() {
58     char ch[80], temp;
59     printf("Enter an expression : ");
60     scanf("%s", ch);
61     if(isBalanced(ch) == 1) {
62         printf("Expression is
balanced.\n");
63     } else {
64         printf("Expression is not
balanced.\n");
65     }
66 }
67

```

Debugger

Sample Test Cases

+

Question Hints

+

< Prev

Reset

Submit

Next >



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.

Explorer

QueueList...

QueueList...

Submit

```

1  typedef struct queue {
2      Q front = NULL, rear = NULL;
3  }
4
5  void enqueue(int element) {
6      Q temp = NULL;
7      temp = (Q)malloc(sizeof(struct
8      queue));
9      if(temp == NULL) {
10         printf("Queue is
11         overflow.\n");
12     } else {
13         temp->data = element;
14         temp->next = NULL;
15         if(front == NULL) {
16             front = temp;
17         } else {
18             rear->next = temp;
19             rear = temp;
20         }
21         printf("Successfully
22         inserted.\n");
23     }
24 }
25
26 void dequeue() {
27     Q temp;
28     if(front == NULL) {
29         printf("Queue is
30         underflow.\n");
31     } else {
32         temp = front;
33         printf("Deleted value =
34         %d\n", temp->data);
35         front = front->next;
36         free(temp);
37         if(front == NULL) {
38             rear = NULL;
39         }
40     }
41 }
42
43 void display() {
44     if(front == NULL) {
45         printf("Queue is empty.\n");
46     } else {
47         Q temp = front;
48         printf("Elements in the
49         queue : ");
50         while(temp != NULL) {
51             printf("%d ", temp->
52             data);
53             temp = temp->next;
54         }
55         printf("\n");
56     }
57 }
58 void size() {
59     Q temp = front;
60     int count = 0;
61     while(temp != NULL) {
62         count++;
63         temp = temp->next;
64     }
65     printf("Queue size :
66     %d\n", count);
67 }
68
69 void isEmpty() {
70     if(front == NULL) {
71         printf("Queue is empty.\n");
72     } else {
73         printf("Queue is not
74         empty.\n");
75     }
76 }

```

Debugger

Plots

Sample Test Cases

+

Question Hints

+

Prev

Reset

Submit

Next



GGU
GURU GURU UNIVERSITY

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.

QueueUser...

QueueOper...

Submit

```

1  struct queue {
2      int data;
3      struct queue *next;
4  };
5
6  typedef struct queue *Q;
7  Q front = NULL, rear = NULL;
8
9  void enqueue(int element) {
10     Q newNode = (Q)malloc(sizeof(struct
11     queue));
12     newNode->data=element;
13     newNode->next=NULL;
14     if(rear==NULL){
15         front=rear=newNode;
16     }
17     else{
18         rear->next=newNode;
19         rear=newNode;
20     }
21     printf("Successfully
22     inserted.\n");
23 }
24
25 void dequeue() {
26     if(front==NULL){
27         printf("Queue is underflow.\n");
28         return;
29     }
30     Q temp=front;
31     printf("Deleted value =
32     %d\n",front->data);
33     front=front->next;
34     free(temp);
35     if(front==NULL){
36         rear=NULL;
37     }
38 }
39
40 void size() {
41     int count=0;
42     Q temp=front;
43     while(temp!=NULL){
44         count++;
45         temp=temp->next;
46     }
47     printf("Queue size :
48     %d\n",count);
49 }
50
51 void isEmpty() {
52     if(front==NULL)
53         printf("Queue is empty.\n");
54     else
55         printf("Queue is not empty.\n");
56 }
57
58 void display() {
59     if(front == NULL) {
60         printf("Queue is empty.\n");
61     } else {
62         Q temp = front;
63         printf("Elements in the
64         queue : ");
65         while(temp != NULL) {
66             printf("%d ", temp ->
67             data);
68             temp = temp -> next;
69         }
70         printf("\n");
71     }

```

Sample Test Cases

+

Prev

Reset

Submit

Next >



GGU

GUJARATI UNIVERSITY

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
```

QueueOper...

QueueOper...

Submit

```

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
15     stack));
16     if(temp == NULL) {
17         printf("Stack is
18         overflow.\n");
19     }
20     else {
21         temp->data = x;
22         temp->next = *stack;
23         *stack = temp;
24     }
25 }
26 int pop(stk* stack) {
27     stk temp;
28     if(*stack == NULL) {
29         printf("Stack is
30         underflow.\n");
31         return -1;
32     } else {
33         temp = *stack;
34         *stack = temp->next;
35         int data=temp->data;
36         free(temp);
37         return data;
38     }
39 }
40 void enqueue(int ele) {
41     push(&(que->stack1),ele);
42     printf("Successfully
43     inserted.\n");
44 }
45 void dequeue() {
46     int x;
47     if(que->stack1==NULL&&que-
48     >stack2==NULL){
49         printf("Queue is
50         underflow.\n");
51         return;
52     }
53     if(que->stack2==NULL){
54         while(que->stack1!=NULL){
55             x=pop(&(que->stack1));
56             push(&(que->stack2),x);
57         }
58         x=pop(&(que->stack2));
59         printf("Deleted value = %d\n",x);
60     }
61 }
62 void display() {
63     stk temp;
64     if(que->stack1==NULL&&que-
65     >stack2==NULL){
66         printf("Queue is empty.\n");
67         return ;
68     }
69     temp=que->stack1;
70     while(temp!=NULL){
71         printf("%d ", temp->data);
72         temp=temp->next;
73     }
74     temp=que->stack2;
75     while(temp!=NULL){
76         printf("%d ", temp->data);
77         temp=temp->next;
78     }
79 }
```

Debugger

Sample Test Cases

+

Prev

Reset

Submit

Next



2.4.2. Implementation and Operations of Circ...

08:40

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 type
// 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`.

NodeCreati...

CreateANo...

Submit

```

1  struct node {
2      int data;
3      struct node *next;
4  };
5  typedef struct node * NODE;
6
7  NODE createNodeInCLL() {
8      NODE temp;
9      temp=(NODE)malloc(sizeof(struct
10     node));
11     if(temp==NULL){
12         printf("memory allocation
13         failed\n");
14         exit(1);
15     }
16     temp->next=temp;
17     return temp;
18 }
19
20 void displayCLL(NODE head){
21     if(head==NULL){
22         printf("List is empty\n");
23         return;
24     }
25     NODE current=head;
26     printf("The list is : ");
27     do{
28         printf("%d -> ", current-
29         >data);
30         current=current->next;
31     }
32     while(current!=head);
33     printf("%d\n", head->data);

```

Debugger

Sample Test Cases

+

< Prev

Reset

Submit

Next >

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**.

Explorer

CircularLL...

CreateAnd...

Submit

```

1  struct node {
2      int data;
3      struct node *next;
4  };
5  typedef struct node *NODE;
6
7  NODE createNodeInCLL() {
8      NODE temp;
9      temp = (NODE)
      malloc(sizeof(struct node));
10     temp->next = NULL;
11     return temp;
12 }
13
14 NODE addNodesInCLL(NODE first, int
x) {
15     NODE newNode =createNodeInCLL();
16     newNode->data=x;
17
18     if(first==NULL){
19         newNode->next=newNode;
20         first=newNode;
21     }
22     else{
23         NODE temp=first;
24         while(temp->next!=first){
25             temp=temp->next;
26         }
27         temp->next=newNode;
28         newNode->next=first;
29     }
30     return first;
31 }
32
33
34
35 void traverseListInCLL(NODE first) {
36
37     if(first==NULL){
38         return;
39     }
40
41     NODE temp=first;
42     do{
43         printf("%d --> ",temp->data);
44         temp=temp->next;
45     }while(temp!=first);
46     printf("\n");
47
48
49
50
51
52

```

Sample Test Cases
+

Question Hints
+

< Prev
Reset
Submit
Next >



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.

```

1  struct node {
2      int data;
3      struct node *next;
4  };
5  typedef struct node *NODE;
6
7  NODE createNodeInCLL() {
8      NODE temp;
9      temp = (NODE)
10     malloc(sizeof(struct node));
11     temp->next = NULL;
12     return temp;
13 }
14
15 NODE insertAtBeginInCLL(NODE first,
16 int x) {
17     NODE temp=
18     (NODE)malloc(sizeof(struct node));
19     temp->data=x;
20     if(first==NULL){
21         temp->next=temp;
22         return temp;
23     }
24     NODE current=first;
25     while(current->next!=first){
26         current=current->next;
27     }
28     current->next=temp;
29     temp->next=first;
30     return temp;
31 }
32
33 int countInCLL(NODE first){
34     if(first==NULL)
35         return 0;
36     int sum=1;
37     NODE temp=first->next;
38     while(temp!=first){
39         sum++;
40         temp=temp->next;
41     }
42     return sum;
43 }
44
45 void traverseListInCLL(NODE first){
46     NODE temp=first;
47     do {
48         printf("%d --> ", temp->
49         data);
50         temp = temp->next;
51     } while (temp != first);
52     printf("\n");
53 }

```

Sample Test Cases



Question Hints





GGU
GUJARATI GATEWAY
UNIVERSITY

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

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

Explorer

CircularLL... InsAtEndin... Submit

```
1 struct node {
2     int data;
3     struct node *next;
4 };
5 typedef struct node *NODE;
6
7 NODE createNodeInCLL() {
8     NODE temp;
9     temp = (NODE)
10    malloc(sizeof(struct node));
11    temp->next = NULL;
12    return temp;
13 }
14
15 NODE insertAtEndInCLL(NODE first,
16 int x) {
17     NODE temp=
18     (NODE)malloc(sizeof(struct node));
19     temp->data=x;
20     temp->next=NULL;
21     if(first==NULL){
22         temp->next=temp;
23         return temp;
24     }
25     NODE current=first;
26     while(current->next!=first){
27         current=current->next;
28     }
29     current->next=temp;
30     temp->next=first;
31     return first;
32 }
33
34 void traverseListInCLL(NODE first) {
35     NODE temp = first;
36     do {
37         printf("%d --> ", temp->
38 data);
39         temp = temp->next;
40     } while (temp != first);
41     printf("\n");
42 }
```

Debugger

Sample Test Cases +

Question Hints +

< Prev

Reset

Submit

Next >



2.4.8. Write a Program to Create a Double Lin... 01:30

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()**.

NodeCreati...

CreateANo...

Submit

1

2

3

4

5

6

7

8

9

10

11

12

13

14

```

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;
    return temp;
}

```

Debugger

Sample Test Cases

+

< Prev

Reset

Submit

Next >



GGU
GURU GURU UNIVERSITY

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.

Explorer

DoubleLL2.c
InsAtBegin...
Submit

```

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 =
11     (NODE)malloc(sizeof(struct node));
12     temp->prev = NULL;
13     temp->next = NULL;
14     return temp;
15 }
16
17 int countInDLL(NODE first) {
18     int count=0;
19     NODE current=first;
20     while(current!=NULL){
21         count++;
22         current=current->next;
23     }
24     return count;
25 }
26
27 NODE insertAtBeginInDLL(NODE first,
28 int x) {
29     NODE temp=
30     (NODE)malloc(sizeof(struct node));
31     if(temp==NULL){
32         printf("Memory allocation
33         failed.\n");
34         return first;
35     }
36     temp->data=x;
37     temp->prev=NULL;
38     temp->next=first;
39     if(first!=NULL){
40         first->prev=temp;
41     }
42     first=temp;
43     return first;
44 }
45
46 void traverseListInDLL(NODE first) {
47     NODE lastNode = first;
48     while (lastNode != NULL) {
49         printf("%d <-> ", lastNode->
50         data);
51         lastNode = lastNode->next;
52     }
53     printf("NULL\n");
54 }

```

Debugger

Sample Test Cases

Question Hints

< Prev

Reset

Submit

Next >



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.

Explorer

DoubleLL4.c

InsAtPositi...

Submit

```

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 =
11     (NODE)malloc(sizeof(struct node));
12     temp->prev = NULL;
13     temp->next = NULL;
14     return temp;
15 }
16
17 NODE insertAtPositionInDLL(NODE
18 first, int position, int x) {
19     NODE temp, current;
20     int i;
21     if(position < 1){
22         printf("Invalid
23         position.\n");
24         return first;
25     }
26     temp = (NODE)malloc(sizeof(struct
27     node));
28     if(temp == NULL){
29         printf("Memory allocation
30         failed.\n");
31         return first;
32     }
33     temp->data = x;
34     temp->prev = NULL;
35     temp->next = NULL;
36     if(position == 1){
37         temp->next = first;
38         if(first != NULL)
39             first->prev = temp;
40         first = temp;
41         return first;
42     }
43     current = first;
44     for(i = 1; i < position -
45     1 && current != NULL; i++){
46         current = current->next;
47     }
48     if(current == NULL){
49         printf("No such position in
50         DLL so insertion is not possible\n");
51         free(temp);
52         return first;
53     }
54     temp->next = current->next;
55     temp->prev = current;
56     if(current->next != NULL)
57         current->next->prev = temp;
58     current->next = temp;
59     return first;
60 }
61
62 void traverseListInDLL(NODE first) {
63     NODE lastNode = first;
64     while (lastNode != NULL) {
65         printf("%d <-> ", lastNode->
66         data);
67         lastNode = lastNode->next;
68     }
69     printf("NULL\n");
70 }

```

Debugger

Sample Test Cases

Question Hints

< Prev

Reset

Submit

Next >