

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT on

Data Structures using C

Submitted by

Sanvi Nadiga (1BM22CS245)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING



B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019

June-2023 to September-2023

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering



CERTIFICATE

This is to certify that the Lab work entitled “**Data Structures using C**” carried out by **SANVI NADIGA (1BM22CS245)**, who is a bonafide student of **B.M.S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the academic semester December-2023 to March-2024. The Lab report has been approved as it satisfies the academic requirements in respect of a **Data Structures using C (23CS3PCDST)** work prescribed for the said degree.

Name of the Lab-In charge: Gauri

Dr. Jyothi S Nayak

(Designation)

Professor and Head

Department of CSE

Department of CSE

BMSCE, Bengaluru

BMSCE, Bengaluru

Index Sheet

Lab Program No.	Program Details	Page No.
1	<p>Write a program to simulate the working of stack using an array with the following : a) Push b) Pop c) Display</p> <p>The program should print appropriate messages for stack overflow, stack underflow</p>	
2	<p>a) WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide)</p> <p>b) Demonstration of account creation on LeetCode platform Program - Leetcode platform</p>	
3	<p>3a) WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display The program should print appropriate messages for queue empty and queue overflow conditions</p> <p>3b) WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display The program should print appropriate messages for queue empty and queue overflow conditions</p>	
4	<p>4a) WAP to Implement Singly Linked List with following operations</p> <p style="padding-left: 40px;">a) Create a linked list.</p> <p style="padding-left: 40px;">b) Insertion of a node at first position, at any position and at end of list.</p> <p>Display the contents of the linked list.</p> <p>4b) Program - Leetcode platform</p>	
5	<p>5a) WAP to Implement Singly Linked List with following operations</p> <p style="padding-left: 40px;">a) Create a linked list.</p> <p style="padding-left: 40px;">b) Deletion of first element, specified element and last element in the list.</p>	

	<p>Display the contents of the linked list.</p> <p>5b) Program - Leetcode platform</p>	
6	<p>6a) WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.</p> <p>6b) WAP to Implement Single Link List to simulate Stack & Queue Operations.</p>	
7	<p>7a) WAP to Implement doubly link list with primitive operations</p> <ol style="list-style-type: none"> Create a doubly linked list. Insert a new node to the left of the node. Delete the node based on a specific value <p>Display the contents of the list</p> <p>7b) Program - Leetcode platform</p>	
8	<p>8a) Write a program</p> <ol style="list-style-type: none"> To construct a binary Search tree. To traverse the tree using all the methods i.e., in-order, preorder and post order <p>To display the elements in the tree.</p> <p>8b) Program - Leetcode platform</p>	
9	<p>9a) Write a program to traverse a graph using BFS method.</p> <p>9b) Write a program to check whether given graph is connected or not using DFS method.</p>	
10	<p>Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F.</p> <p>Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT.</p> <p>Let the keys in K and addresses in L are integers.</p> <p>Design and develop a Program in C that uses Hash function $H: K \rightarrow L$ as $H(K)=K \bmod m$ (remainder method), and implement hashing technique to map a given key K to the address space L.</p> <p>Resolve the collision (if any) using linear probing.</p>	

Course Outcome

CO1	Apply the concept of linear and nonlinear data structures.
CO2	Analyse data structure operations for a given problem.
CO3	CO3 Design and implement operations of linear and nonlinear data structure.
CO4	Conduct practical experiments for demonstrating the operations of different data structures and sorting techniques.

Lab Program 1

```
//Program to simulate the working of stack using an array with the push,pull and
display functions- #include <stdio.h>
#include <stdlib.h>
#define N 5

int stack[N]; int top=-1;

void main() {

int choice;
do
{
printf("\nMenu\n");
printf("1.Push 2.Pop 3. Display 4.Exit\n"); printf("Enter your choice\n");
scanf("%d",&choice);

switch (choice) {
case 1:

push();

break; case 2:

pop(); break;

case 3: display();

break; case 4:

exit(0);

break; default:

printf("Invalid input"); }

}while(choice!=4); }

void push() {
```

```
int x; if(top==(N-1)) {  
  
printf("Stack is full,Overflow condition\n");  
  
return; }  
  
else {  
  
printf("Enter element to be inserted\n"); scanf("%d",&x);  
top++;  
stack[top]=x;  
  
printf("Element inserted is=%d\n",stack[top]); }  
  
}  
  
void pop() {  
  
int item;  
if (top==-1) {  
  
printf("Stack is empty ,underflow condition\n");  
  
return; }  
  
else {  
  
}}  
  
void display() {  
  
printf("Elements in the stack are:\n"); for (int i=top;i>=0;i--)  
  
item=stack[top];  
top--;  
printf("Element deleted is =%d",item);  
  
printf("%d\n",stack[i]);  
  
}
```

OUTPUT:

```
Menu
1.Push 2.Pop 3. Display 4.Exit
Enter your choice
1
Enter element to be inserted
10
Element inserted is=10

Menu
1.Push 2.Pop 3. Display 4.Exit
Enter your choice
1
Enter element to be inserted
20
Element inserted is=20

Menu
1.Push 2.Pop 3. Display 4.Exit
Enter your choice
1
Enter element to be inserted
30
Element inserted is=30

Menu
1.Push 2.Pop 3. Display 4.Exit
Enter your choice
1
Enter element to be inserted
40
Element inserted is=40

Menu
1.Push 2.Pop 3. Display 4.Exit
Enter your choice
1
Enter element to be inserted
50
Element inserted is=50

Menu
1.Push 2.Pop 3. Display 4.Exit
Enter your choice
```



```
Menu
1.Push 2.Pop 3. Display 4.Exit
Enter your choice
1
Enter element to be inserted
30
Element inserted is=30

Menu
1.Push 2.Pop 3. Display 4.Exit
Enter your choice
1
Enter element to be inserted
40
Element inserted is=40

Menu
1.Push 2.Pop 3. Display 4.Exit
Enter your choice
1
Enter element to be inserted
50
Element inserted is=50

Menu
1.Push 2.Pop 3. Display 4.Exit
Enter your choice
1
Stack is full,Overflow condition

Menu
1.Push 2.Pop 3. Display 4.Exit
Enter your choice
3
Elements in the stack are:
50
40
30
20
10
```

Lab Program-2

INFIX TO POSTFIX:

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

```
#define MAX 100
```

```
char stack[MAX];
char infix[MAX], postfix[MAX];
int top = -1;

void push(char);
char pop();
int isempty();
void infixToPostfix();
int space(char);
int print();
int precedence(char);

int main()
{
    printf("enter infix exp: \n");
    gets(infix);

    infixToPostfix();
    print();
    return 0;
}

void infixToPostfix()
{
    int i, j = 0;
    char next, symbol;
    for (i = 0; i < strlen(infix); i++)
    {
        symbol = infix[i];
        if (!space(symbol))
        {
            switch (symbol)
```

```

{
case '(':
    push(symbol);
    break;

case ')':
    while ((next = pop()) != '(')
    {
        postfix[j++] = next;
    }
    break;

case '+':
case '-':
case '*':
case '/':
case '^':
    while (!isempty() && precedence(stack[top]) >= precedence(symbol))
    {
        postfix[j++] = pop();
    }
    push(symbol);
    break;
default:
    postfix[j++] = symbol;
}
}

while (!isempty())
{
    postfix[j++] = pop();
}

```

```
    postfix[j] = '\0';
}

int space(char c)
{
    if (c == ' ' || c == '\t')
    {
        return 1;
    }
    else
        return 0;
}

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

int print()
{
    int i = 0;
```

```
printf("postfix exp: \n");
while (postfix[i])
{
    printf("%c", postfix[i++]);
}
printf("\n");
}

void push(char c)
{
    if (top == MAX - 1)
    {
        printf("stack overflow\n");
        return;
    }
    else
    {
        top++;
        stack[top] = c;
    }
}

char pop()
{
    char c;
    if (top == -1)
    {
        printf("stack underflow\n");
        exit(1);
    }
    else
    {
        c = stack[top];
```

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

output:

```
C:\Users\bmsce\Desktop\1BM X + v  
enter infix exp:  
(A+B)/(C+D)-(D*E)  
postfix exp:  
AB+CD+/DE*-  
  
Process returned 0 (0x0) execution time : 40.464 s  
Press any key to continue.
```

Lab Program-2b

Code:

```
#include <stdbool.h>
```

```
#include <stdlib.h>
```

```
typedef struct {
```

```
    int* data;
```

```
    int front;
```

```
    int rear;
```

```
    int size;
```

```
} Queue;
```

```
typedef struct {
```

```
    Queue* q1;
```

```
    Queue* q2;
```

```
} MyStack;
```

```
Queue* createQueue(int size) {
```

```
    Queue* queue = (Queue*)malloc(sizeof(Queue));
```

```
    queue->data = (int*)malloc(size * sizeof(int));
```

```
    queue->front = queue->rear = -1;
```

```
    queue->size = size;
```

```
    return queue;
```

```
}
```

```
void enqueue(Queue* queue, int value) {
```

```
    if (queue->rear == -1) {
```

```
        queue->front = queue->rear = 0;
```

```
    } else {
```

```
        queue->rear = (queue->rear + 1) % queue->size;
```

```
    }
```

```
    queue->data[queue->rear] = value;
```

```
}
```

```

int dequeue(Queue* queue) {
    int value = queue->data[queue->front];
    if (queue->front == queue->rear) {
        queue->front = queue->rear = -1;
    } else {
        queue->front = (queue->front + 1) % queue->size;
    }
    return value;
}

```

```

bool isEmpty(Queue* queue) {
    return queue->front == -1;
}

```

```

MyStack* myStackCreate() {
    MyStack* stack = (MyStack*)malloc(sizeof(MyStack));
    stack->q1 = createQueue(1000); // Adjust the size as needed
    stack->q2 = createQueue(1000);
    return stack;
}

```

```

void myStackPush(MyStack* obj, int x) {
    enqueue(obj->q1, x);
}

```

```

int myStackPop(MyStack* obj) {
    if (isEmpty(obj->q1)) {
        return -1; // Stack is empty
    }

    while (obj->q1->front != obj->q1->rear) {
        enqueue(obj->q2, dequeue(obj->q1));
    }
}

```



```

int poppedValue = dequeue(obj->q1);

// Swap q1 and q2
Queue* temp = obj->q1;
obj->q1 = obj->q2;
obj->q2 = temp;

return poppedValue;
}

int myStackTop(MyStack* obj) {
    if (isEmpty(obj->q1)) {
        return -1; // Stack is empty
    }

    while (obj->q1->front != obj->q1->rear) {
        enqueue(obj->q2, dequeue(obj->q1));
    }

    int topValue = dequeue(obj->q1);
    enqueue(obj->q2, topValue);

    // Swap q1 and q2
    Queue* temp = obj->q1;
    obj->q1 = obj->q2;
    obj->q2 = temp;

    return topValue;
}

bool myStackEmpty(MyStack* obj) {
    return isEmpty(obj->q1);
}

```

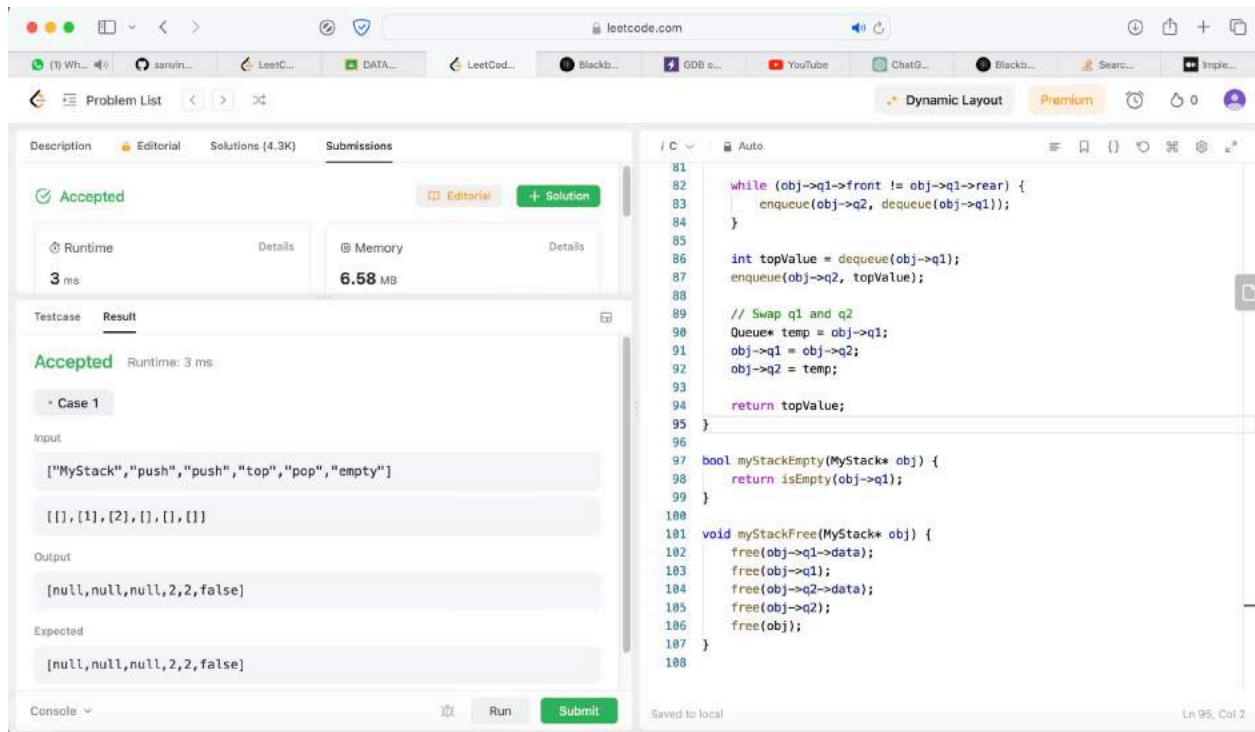
```

}

void myStackFree(MyStack* obj) {
    free(obj->q1->data);
    free(obj->q1);
    free(obj->q2->data);
    free(obj->q2);
    free(obj);
}

```

Output:



Lab Program-3a

ARRAY IMPLEMENTATION IN QUEUE:

```
#include <stdio.h> #define MAX 100
int queue_array[MAX]; int rear=-1;
int front=-1;

int display() {
    int i; if(front== -1) {
        printf("queue is empty\n"); }
    else {
        printf("queue is: \n"); for(i=front; i<=rear; i++) {
            printf("%d", queue_array[i]); }
        printf("\n"); }
    }

int main() {
    int choice; while(1)
    {
        printf("1. insert \n"); printf("2. delete \n"); printf("3. display \n"); printf("4. exit \n");
        printf("enter choice: \n"); scanf("%d", &choice); switch(choice)
        {
            case 1:
                insert();
```

```
break; case 2:

delete();

break; case 3:

display(); break;

default:
printf("invalid input\n"); break;

}}

}

int insert() {

int add_item; if(rear==MAX-1) {

printf("queue overflow\n"); }

else {

if(front==MAX-1) {

front=0; }

printf("insert element: \n"); scanf("%d",&add_item); rear++;
queue_array[rear]=add_item;

}}

int delete() {

if (front==MAX-1 || front>rear) {

printf("queue overflow\n");

return; }

else {
```

```
printf("delete element: \n"); scanf("%d",&queue_array[front]); front++;
}}
```

OUTPUT:

```
4. exit  
enter choice:  
1  
insert element:  
2  
1. insert  
2. delete  
3. display  
4. exit  
enter choice:  
2  
delete element:  
1  
1. insert  
2. delete  
3. display  
4. exit  
enter choice:  
3  
queue is:  
  
1. insert  
2. delete  
3. display  
4. exit  
enter choice:  
1  
insert element:
```

```
queue is:  
  
1. insert  
2. delete  
3. display  
4. exit  
enter choice:  
1  
insert element:  
3  
1. insert  
2. delete  
3. display  
4. exit  
enter choice:  
1  
insert element:  
5  
1. insert  
2. delete  
3. display  
4. exit  
enter choice:  
3  
queue is:  
35  
1. insert
```

Lab Program-3b

CIRCULAR QUEUE:

```
#include <stdio.h>
```

```
# define max 6
```

```
int queue[max]; // array declaration
```

```
int front=-1;
```

```
int rear=-1;
```

```
// function to insert an element in a circular queue
```

```
void enqueue(int element)
```

```
{
```

```
    if(front==-1 && rear==-1) // condition to check queue is empty
```

```
    {
```

```
        front=0;
```

```
        rear=0;
```

```
        queue[rear]=element;
```

```
    }
```

```
    else if((rear+1)%max==front) // condition to check queue is full
```

```
    {
```

```
        printf("Queue is overflow..");
```

```
    }
```

```
    else
```

```
    {
```

```
        rear=(rear+1)%max;    // rear is incremented
```

```
        queue[rear]=element;    // assigning a value to the queue at the rear
```

```
position.
```

```
    }
```

```
}
```

```
// function to delete the element from the queue
```

```

int dequeue()
{
    if((front== -1) && (rear== -1)) // condition to check queue is empty
    {
        printf("\nQueue is underflow..");
    }
    else if(front==rear)
    {
        printf("\nThe dequeued element is %d", queue[front]);
        front=-1;
        rear=-1;
    }
    else
    {
        printf("\nThe dequeued element is %d", queue[front]);
        front=(front+1)%max;
    }
}

// function to display the elements of a queue
void display()
{
    int i=front;
    if(front== -1 && rear== -1)
    {
        printf("\n Queue is empty..");
    }
    else
    {
        printf("\nElements in a Queue are :");
        while(i<=rear)
        {
            printf("%d,", queue[i]);
            i=(i+1)%max;
        }
    }
}

```

```

    }
}
int main()
{
    int choice=1,x; // variables declaration

    while(choice<4 && choice!=0) // while loop
    {
        printf("\n Press 1: Insert an element");
        printf("\nPress 2: Delete an element");
        printf("\nPress 3: Display the element");
        printf("\nEnter your choice");
        scanf("%d", &choice);

        switch(choice)
        {

            case 1:

                printf("Enter the element which is to be inserted");
                scanf("%d", &x);
                enqueue(x);
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();

        }}
    return 0;
}

```


OUTPUT:

```

Press 3: Display the element
Enter your choice
1
Enter the element which is to be inserted2

Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice1
Enter the element which is to be inserted3

Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice1
Enter the element which is to be inserted4

Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice3

Elements in a Queue are :2,3,4,
Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice2

The dequeued element is 2
Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice3

Elements in a Queue are :3,4,
Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice1
Enter the element which is to be inserted9

Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice3

Elements in a Queue are :3,4,9,
Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice

```

Lab Program-4a(a and b)

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

typedef struct Node {
    int data;
    struct Node *next;
}Node;

void InsertAtBeginning( Node **head_ref,int new_data);
void InsertAtEnd( Node **head_ref,int new_data);
void Insert( Node **prev_node,int new_data,int pos);
void PrintList(Node * next);

void InsertAtBeginning( Node **head_ref,int new_data)
{
    Node *new_node=(struct Node*)malloc(sizeof( Node));
    new_node->data=new_data;
    new_node->next=*head_ref;
    *head_ref=new_node;
}

void InsertAtEnd(Node **head_ref,int new_data)
{
    Node *new_node=(struct Node*)malloc(sizeof( Node));
    Node *last=*head_ref;
    new_node->data=new_data;
    new_node->next=NULL;
```

```
if (*head_ref==NULL)
{
    *head_ref=new_node;
    return ;
}
while (last->next!=NULL)
    last=last->next;
last->next=new_node;

}

void Insert(Node **head_ref,int new_data,int pos)
{
    if (*head_ref ==NULL)
    {
        printf("Cannot be NULL\n");
        return;
    }
    Node *temp = *head_ref;
    Node *newNode = ( Node *) malloc (sizeof ( Node));
    newNode->data = new_data;
    newNode->next = NULL;

    while (--pos>0)
    {
        temp = temp->next;
    }
    newNode->next = temp->next;
    temp->next = newNode;
}

void PrintList(Node *node)
```

```
{
    while (node!=NULL)
    {
        printf("%d\n",node->data);
        node=node->next;
    }
}
```

```
int main()
{
    int ch,new,pos;
    Node* head=NULL;
    while(ch!=5)
    {
        printf("Menu\n");
        printf("1.Insert at beginning\n");
        printf("2.Insert at a specific position\n");
        printf("3.Insert at end\n");
        printf("4.Display linked list\n");
        printf("5.Exit\n");
        printf("Enter your choice\n");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
            {
                printf("Enter the data you want to insert at beginning\n");
                scanf("%d",&new);
                InsertAtBeginning(&head,new);
                break;
            }
            case 2:
```

```
{
printf("Enter the data and position at which you want to insert \n");
scanf("%d%d",&new,&pos);
Insert(&head,new,pos);
break;
}
case 3:
{
printf("Enter the data you want to insert at end\n");
scanf("%d",&new);
InsertAtEnd(&head,new);
break;
}
case 4:
{
printf("Created linked list is:\n");
PrintList(head);
break;
}
case 5:
{
return 0;
break;
}
case 6:
{
printf("Invalid data!");
break;
}
}
}
```

return 0;

Output:

```
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
1
Enter the data you want to insert at beginning
23
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
1
Enter the data you want to insert at beginning
54
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
1
Enter the data you want to insert at beginning
55
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
4
Created linked list is:
55
54
23
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
```

Lab Program-4b

```

/**
 * Definition for singly-linked list. * struct ListNode {
 * int val;
 * struct ListNode *next;
 * };
 */
#include <stdlib.h>
#include <stdio.h>

// Definition for singly-linked list.

struct ListNode* reverseList(struct ListNode* head) { struct ListNode* prev =
NULL;
struct ListNode* curr = head;
struct ListNode* next = NULL;

while (curr != NULL) {
next = curr->next; // Save the next node curr->next = prev; // Reverse the link
prev = curr; // Move prev to the current node curr = next; // Move curr to the next
node
}

return prev; // The new head of the reversed list }

// Function to print the linked list

void printList(struct ListNode* head) { while (head != NULL) {
printf("%d -> ", head->val);
head = head->next;

} printf("NULL\n"); }

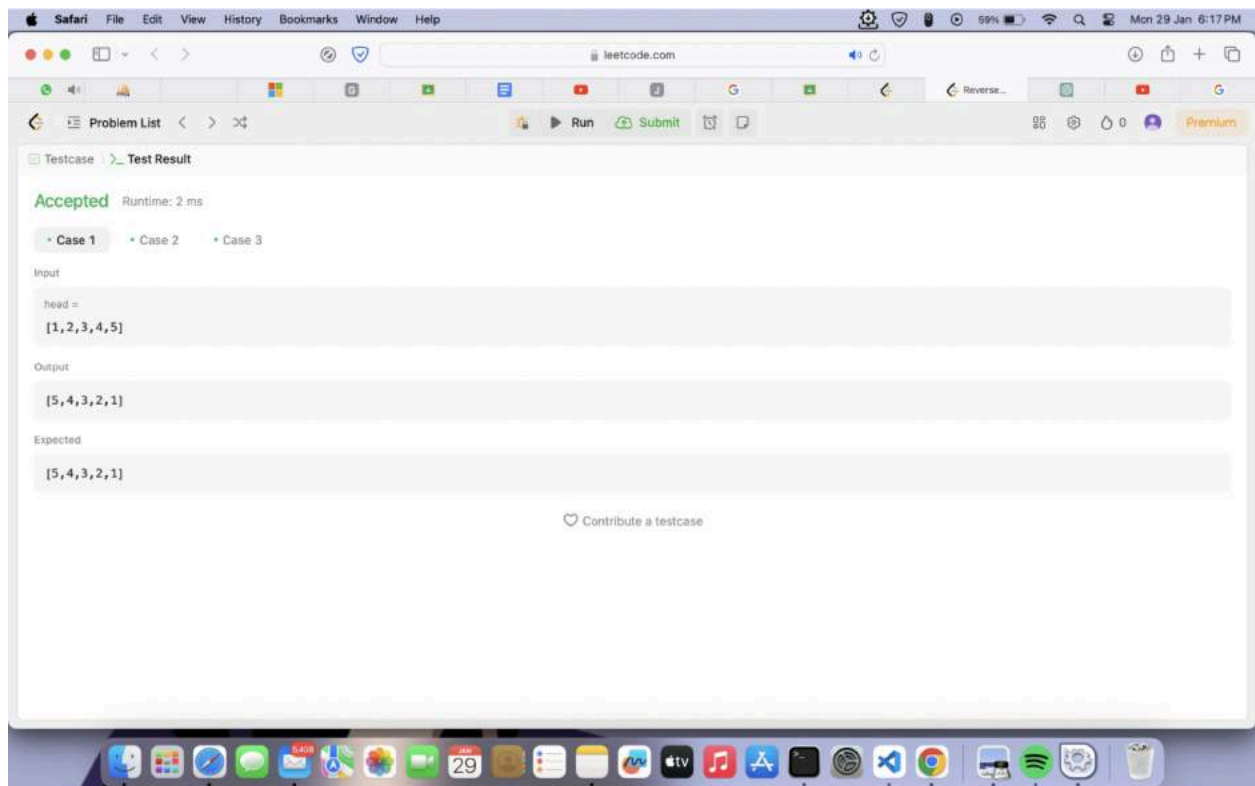
// Function to create a new node with a given value

```

Data Structures using C(23CS3PCDST)

```
struct ListNode* createNode(int val) {  
    struct ListNode* newNode = (struct ListNode*)malloc(sizeof(struct ListNode));  
    newNode->val = val;  
    newNode->next = NULL;  
    return newNode;  
}
```

Output:



Lab Program-5

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

typedef struct Node {
    int data;
    struct Node *next;
}Node;

void InsertAtBeginning( Node **head_ref,int new_data);
void DeleteAtBeginning( Node **head_ref);
void DeleteAtEnd( Node **head_ref);
void Delete( Node **prev_node,int pos);
void PrintList(Node * next);

void InsertAtBeginning( Node **head_ref,int new_data)
{
    Node *new_node=(struct Node*)malloc(sizeof( Node));
    new_node->data=new_data;
    new_node->next=*head_ref;
    *head_ref=new_node;
}

void DeleteAtBeginning( Node **head_ref)
{
    Node *ptr;
    if(head_ref == NULL)
    {
        printf("\nList is empty");
    }
    else
```

```
{  
ptr = *head_ref;  
*head_ref = ptr->next;  
free(ptr);  
printf("\n Node deleted from the beginning ...");  
  
}  
  
}
```

```
void DeleteAtEnd(Node **head_ref)  
{  
    Node *ptr,*ptr1;  
  
    if(*head_ref == NULL)  
  
    {  
  
        printf("\nlist is empty");  
  
    }  
  
    else if((*head_ref)-> next == NULL)  
  
    {  
  
        free(*head_ref);  
  
        *head_ref= NULL;  
  
        printf("\nOnly node of the list deleted ...");  

```

```
}

else

{

ptr = *head_ref;

while(ptr->next != NULL)

{

ptr1 = ptr;

ptr = ptr ->next;

}

ptr1->next = NULL;

free(ptr);

printf("\n Deleted Node from the last ...");

}

}

void Delete(Node **head_ref, int pos)
{
    Node *temp = *head_ref, *prev;

    if (temp == NULL)
    {
```

```

    printf("\nList is empty");
    return;
}

if (pos == 0)
{
    *head_ref = temp->next;
    free(temp);
    printf("\nDeleted node with position %d", pos);
    return;
}

for (int i = 0; temp != NULL && i < pos - 1; i++)
{
    prev = temp;
    temp = temp->next;
}

if (temp == NULL)
{
    printf("\nPosition out of range");
    return;
}

prev->next = temp->next;
free(temp);
printf("\nDeleted node with position %d", pos);
}

void PrintList(Node *node)
{
    while (node!=NULL)
    {
        printf("%d\n",node->data);

```

```
    node=node->next;
}
}
```

```
int main()
{
    int ch,new,pos;
    Node* head=NULL;
    while(ch!=6)
    {
        printf("Menu\n");
        printf("1.Create a linked list\n");
        printf("2.Delete at beginning\n");
        printf("3.Delete at a specific position\n");
        printf("4..Delete at end\n");
        printf("5..Display linked list\n");
        printf("6..Exit\n");
        printf("Enter your choice\n");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
            {
                printf("Enter the data you want to insert at beginning\n");
                scanf("%d",&new);
                InsertAtBeginning(&head,new);
                break;
            }
            case 2:
            {
                DeleteAtBeginning(&head);
                break;
            }
        }
    }
}
```

```
}
case 3:
{
printf("Enter the position at which you want to delete \n");
scanf("%d",&pos);
Delete(&head,pos);
break;
}
case 4:
{
DeleteAtEnd(&head);
break;
}
case 5:
{
printf("Created linked list is:\n");
PrintList(head);
break;
}
case 6:
{
return 0;
break;
}
default:
{
printf("Invalid data!");
break;
}
}
return 0;
}
```

output:

```

Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
1
Enter the data you want to insert at beginning
32
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
1
Enter the data you want to insert at beginning
43
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
1
Enter the data you want to insert at beginning
54
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
5
Created linked list is:
54
43
32
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end

```

```

Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
2

Node deleted from the beginning ...Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
5
Created linked list is:
43
32
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
1
Enter the data you want to insert at beginning
23
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
3
Enter the position at which you want to delete
3

Deleted node with position 3Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end

```



```

2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
3
Enter the position at which you want to delete
3

Deleted node with position 3Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
5
Created linked list is:
23
43
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
4

Deleted Node from the last ...Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
5
Created linked list is:
23
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice

```

Lab Program-6a

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

struct Node {
    int data;
    struct Node* next;
};

typedef struct Node Node;

Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

void append(Node** head, int data) {
    Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        Node* current = *head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newNode;
    }
}

void display(Node* head) {
```

```

Node* current = head;
while (current != NULL) {
    printf("%d -> ", current->data);
    current = current->next;
}
printf("NULL\n");
}

void sortList(Node** head) {
    if (*head == NULL) {
        return;
    }

    int temp;
    Node* current1 = *head;
    Node* current2;

    while (current1 != NULL) {
        current2 = current1->next;

        while (current2 != NULL) {
            if (current1->data > current2->data) {
                temp = current1->data;
                current1->data = current2->data;
                current2->data = temp;
            }

            current2 = current2->next;
        }

        current1 = current1->next;
    }
}

```

```

void reverseList(Node** head) {
    Node* prev = NULL;
    Node* current = *head;
    Node* nextNode;

    while (current != NULL) {
        nextNode = current->next;
        current->next = prev;
        prev = current;
        current = nextNode;
    }

    *head = prev;
}

void concatenateLists(Node** list1, Node* list2) {
    if (*list1 == NULL) {
        *list1 = list2;
    } else {
        Node* current = *list1;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = list2;
    }
}

int main() {
    Node* list1 = NULL;
    Node* list2 = NULL;

    append(&list1, 3);

```

```
append(&list1, 1);
append(&list1, 4);

append(&list2, 2);
append(&list2, 5);

printf("Original List 1:\n");
display(list1);

printf("\nSorting List 1:\n");
sortList(&list1);
display(list1);

printf("\nReversing List 1:\n");
reverseList(&list1);
display(list1);

printf("\nOriginal List 2:\n");
display(list2);

printf("\nConcatenating List 1 and List 2:\n");
concatenateLists(&list1, list2);
display(list1);

return 0;
}
```

Output:

```
Original List 1:
3 -> 1 -> 4 -> NULL

Sorting List 1:
1 -> 3 -> 4 -> NULL

Reversing List 1:
4 -> 3 -> 1 -> NULL

Original List 2:
2 -> 5 -> NULL

Concatenating List 1 and List 2:
4 -> 3 -> 1 -> 2 -> 5 -> NULL

Process returned 0 (0x0)   execution time : 0.059 s
Press any key to continue.
```

Lab Program-6b

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

// Define a Node structure for the linked list
typedef struct Node {
    int data;
    struct Node* next;
} Node;

// Function prototypes
Node* createNode(int data);
Node* push(Node* top, int data);
```

```
Node* pop(Node* top, int* poppedValue);
Node* enqueue(Node* rear, int data);
Node* dequeue(Node* front, int* dequeuedValue);
void display(Node* head);
```

```
int main() {
    Node* stackTop = NULL;
    Node* queueFront = NULL;
    Node* queueRear = NULL;
    int choice, data, poppedValue, dequeuedValue;

    do {
        printf("\nMenu:\n");
        printf("1. Push (Stack)\n");
        printf("2. Pop (Stack)\n");
        printf("3. Enqueue (Queue)\n");
        printf("4. Dequeue (Queue)\n");
        printf("5. Display\n");
        printf("6. Exit\n");

        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to push: ");
                scanf("%d", &data);
                stackTop = push(stackTop, data);
                break;

            case 2:
                stackTop = pop(stackTop, &poppedValue);
                if (stackTop != NULL) {
```

```
    printf("Popped element: %d\n", poppedValue);  
}  
break;
```

case 3:

```
    printf("Enter data to enqueue: ");  
    scanf("%d", &data);  
    queueRear = enqueue(queueRear, data);  
    if (queueFront == NULL) {  
        queueFront = queueRear;  
    }  
    break;
```

case 4:

```
    queueFront = dequeue(queueFront, &dequeuedValue);  
    if (queueFront != NULL) {  
        printf("Dequeued element: %d\n", dequeuedValue);  
        if (queueFront == NULL) {  
            queueRear = NULL; // Reset rear if queue becomes empty  
        }  
    }  
    break;
```

case 5:

```
    display(queueFront);  
    break;
```

case 6:

```
    printf("Exiting the program.\n");  
    break;
```

default:

```
    printf("Invalid choice. Please enter a valid option.\n");
```



```

    }

    } while (choice != 6);

    return 0;
}

// Function to create a new node with the given data
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(EXIT_FAILURE);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to push an element onto the stack
Node* push(Node* top, int data) {
    Node* newNode = createNode(data);
    newNode->next = top;
    return newNode;
}

// Function to pop an element from the stack
Node* pop(Node* top, int* poppedValue) {
    if (top == NULL) {
        printf("Stack underflow. Cannot pop from an empty stack.\n");
        return NULL;
    }
    *poppedValue = top->data;

```

```

Node* temp = top;
top = top->next;
free(temp);
return top;
}

```

// Function to enqueue an element into the queue

```

Node* enqueue(Node* rear, int data) {
    Node* newNode = createNode(data);
    if (rear == NULL) {
        // If the queue is empty, set both front and rear to the new node
        return newNode;
    }
    rear->next = newNode;
    return newNode;
}

```

// Function to dequeue an element from the queue

```

Node* dequeue(Node* front, int* dequeuedValue) {
    if (front == NULL) {
        printf("Queue underflow. Cannot dequeue from an empty queue.\n");
        return NULL;
    }
    *dequeuedValue = front->data;
    Node* temp = front;
    front = front->next;
    free(temp);
    return front;
}

```

// Function to display the elements of the linked list

```

void display(Node* head) {
    if (head == NULL) {

```

```
    printf("The list is empty.\n");  
    return;  
}  
  
printf("List elements: ");  
while (head != NULL) {  
    printf("%d ", head->data);  
    head = head->next;  
}  
printf("\n");  
}
```

Output:

```
Menu:
1. Push (Stack)
2. Pop (Stack)
3. Enqueue (Queue)
4. Dequeue (Queue)
5. Display
6. Exit
Enter your choice: 3
Enter data to enqueue: 33
```

```
Menu:
1. Push (Stack)
2. Pop (Stack)
3. Enqueue (Queue)
4. Dequeue (Queue)
5. Display
6. Exit
Enter your choice: 3
Enter data to enqueue: 44
```

```
Menu:
1. Push (Stack)
2. Pop (Stack)
3. Enqueue (Queue)
4. Dequeue (Queue)
5. Display
6. Exit
Enter your choice: 3
Enter data to enqueue: 55
```

```
Menu:
1. Push (Stack)
2. Pop (Stack)
3. Enqueue (Queue)
4. Dequeue (Queue)
5. Display
6. Exit
Enter your choice: 5
List elements: 33 44 55
```

```
Menu:
1. Push (Stack)
2. Pop (Stack)
3. Enqueue (Queue)
4. Dequeue (Queue)
5. Display
6. Exit
Enter your choice: 4
Dequeued element: 33
```

```
Menu:
1. Push (Stack)
2. Pop (Stack)
3. Enqueue (Queue)
4. Dequeue (Queue)
5. Display
6. Exit
Enter your choice: 5
List elements: 33 44 55
```

```
Menu:
1. Push (Stack)
2. Pop (Stack)
3. Enqueue (Queue)
4. Dequeue (Queue)
5. Display
6. Exit
Enter your choice: 4
Dequeued element: 33
```

```
Menu:
1. Push (Stack)
2. Pop (Stack)
3. Enqueue (Queue)
4. Dequeue (Queue)
5. Display
6. Exit
Enter your choice: 33
Invalid choice. Please enter a valid option.
```

```
Menu:
1. Push (Stack)
2. Pop (Stack)
3. Enqueue (Queue)
4. Dequeue (Queue)
5. Display
6. Exit
Enter your choice: 5
List elements: 44 55
```

```
Menu:
1. Push (Stack)
2. Pop (Stack)
3. Enqueue (Queue)
4. Dequeue (Queue)
5. Display
6. Exit
Enter your choice:
```

Lab Program-7a

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

struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(EXIT_FAILURE);
    }
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = createNode(data);

    if (*head == NULL) {
        *head = newNode;
    } else {
        newNode->next = *head;
        (*head)->prev = newNode;
        *head = newNode;
    }
}
```

```

}
void insertBeforeNode(struct Node** head, int key, int data) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }

    struct Node* newNode = createNode(data);
    struct Node* current = *head;

    while (current) {
        if (current->data == key) {
            if (current->prev) {
                current->prev->next = newNode;
                newNode->prev = current->prev;
            } else {
                *head = newNode;
            }

            newNode->next = current;
            current->prev = newNode;
            return;
        }
        current = current->next;
    }

    printf("Key not found in the list\n");
}

```

```

void deleteNode(struct Node** head, int pos) {
    if (*head == NULL) {
        printf("List is empty\n");
    }
}

```

```
    return;
}

struct Node* current = *head;
int count = 1;

while (current && count < pos) {
    current = current->next;
    count++;
}

if (current == NULL) {
    printf("Position %d is beyond the length of the list\n", pos);
    return;
}

if (current->prev) {
    current->prev->next = current->next;
} else {
    *head = current->next;
}

if (current->next) {
    current->next->prev = current->prev;
}

free(current);
printf("Node at position %d deleted\n", pos);
}

void displayList(struct Node* head) {
    if (head == NULL) {
        printf("List is empty\n");
    }
}
```



```

    return;
}

struct Node* current = head;

while (current) {
    printf("%d-> ", current->data);
    current = current->next;
}
printf("\n");
}

void freeList(struct Node* head) {
    struct Node* current = head;
    struct Node* nextNode;

    while (current) {
        nextNode = current->next;
        free(current);
        current = nextNode;
    }
}

int main() {
    struct Node* head = NULL;
    int ch, newData, pos, key;

    while (1) {
        printf("\nMenu\n");
        printf("1. Insert at the beginning\n");
        printf("2. Insert before a node\n");
        printf("3. Delete a node\n");
        printf("4. Display list\n");
    }
}

```

```
printf("5. exit\n");
printf("Enter your choice: ");
scanf("%d", &ch);

switch (ch) {
    case 1:
        printf("Enter data to insert at the beginning: ");
        scanf("%d", &newData);
        insertAtBeginning(&head, newData);
        break;

    case 2:
        printf("Enter the value before which you want to insert: ");
        scanf("%d", &key);
        printf("Enter data to insert: ");
        scanf("%d", &newData);
        insertBeforeNode(&head, key, newData);
        break;

    case 3:
        printf("Enter the position you wish to delete: ");
        scanf("%d", &key);
        deleteNode(&head, key);
        break;

    case 4:
        printf("Doubly linked list: ");
        displayList(head);
        break;

    case 5:
        freeList(head);
        printf("Exiting the program\n");
```

```
        return 0;

    default:
        printf("Invalid choice\n");
    }
}

return 0;
}
```

output:

```
Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. exit
Enter your choice: 1
Enter data to insert at the beginning: 21
```

```
Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. exit
Enter your choice: 1
Enter data to insert at the beginning: 32
```

```
Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. exit
Enter your choice: 1
Enter data to insert at the beginning: 43
```

```
Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. exit
Enter your choice: 1
Enter data to insert at the beginning: 33
```

```
Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. exit
Enter your choice: 4
Doubly linked list: 33-> 43-> 32-> 21->
```

```
Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
```

Menu

1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. exit

Enter your choice: 2

Enter the value before which you want to insert: 11

Enter data to insert:

2

Key not found in the list

Menu

1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. exit

Enter your choice: 2

Enter the value before which you want to insert: 32

Enter data to insert: 55

Menu

1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. exit

Enter your choice: 4

Doubly linked list: 33-> 43-> 55-> 32-> 21->

Menu

1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. exit

Enter your choice: 3

Enter the position you wish to delete: 2

Node at position 2 deleted

Menu

1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. exit

Enter your choice: 4

Doubly linked list: 33-> 55-> 32-> 21->

Lab Program-8

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

```
struct node{
    int data;
    struct node *right_child;
    struct node *left_child;
};
```

```
struct node *new_node(int x){
    struct node* temp;
    temp=(struct node*)malloc(sizeof(struct node));
    temp->data=x;
    temp->right_child=NULL;
    temp->left_child=NULL;
    return temp;
};
```

```
struct node *insert (struct node *root, int x){
    if(root==NULL){
        return new_node(x);
    }
    else if(x > root->data){
        root->right_child=insert(root->right_child,x);
    }
    else{
        root->left_child=insert(root->left_child,x);
    }
}
```

```

    }
    return root;
};

```

```

void preorder(struct node *root){
    if (root!=NULL){
        printf("%d\n", root->data);
        preorder(root->left_child);
        preorder(root->right_child);
    }
};

```

```

void inorder(struct node *root){
    if (root!=NULL){
        inorder(root->left_child);
        printf("%d\n", root->data);
        inorder(root->right_child);
    }
}

```

```

void postorder(struct node *root){
    if (root!=NULL){
        postorder(root->left_child);
        postorder(root->right_child);
        printf("%d\n", root->data);
    }
};

```

```

int main(){
    struct node *root=new_node(100);
    insert(root, 5);
}

```

```
insert(root, 15);
insert(root, 2);
insert(root, 4);
insert(root, 30);
insert(root, 7);
insert(root, 1);

printf("preorder traversal\n");
preorder(root);
printf("\n");
printf("inorder traversal\n");
inorder(root);
printf("\n");
printf("postorder traversal\n");
postorder(root);
printf("\n");
}
```

Output:


```
preorder traversal
```

```
100
```

```
5
```

```
2
```

```
1
```

```
4
```

```
15
```

```
7
```

```
30
```

```
inorder traversal
```

```
1
```

```
2
```

```
4
```

```
5
```

```
7
```

```
15
```

```
30
```

```
100
```

```
postorder traversal
```

```
1
```

```
4
```

```
2
```

```
7
```

```
30
```

```
15
```

```
5
```

```
100
```

```
Process returned 0 (0x0)   execution time : 0.089 s
```

```
Press any key to continue.
```

Lab Program-8b

226. Invert Binary Tree

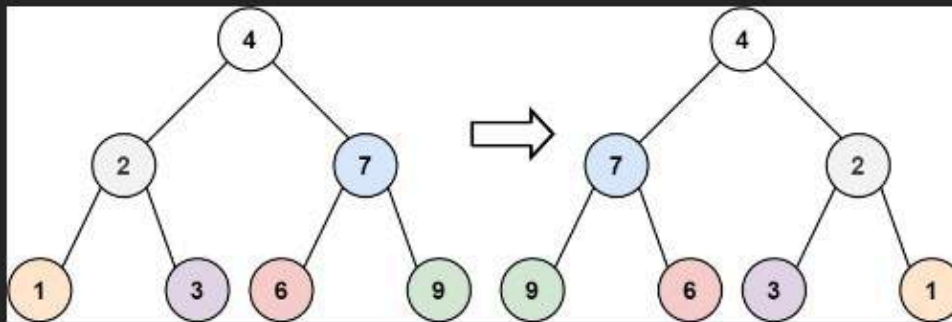
Easy

Topics

Companies

Given the `root` of a binary tree, invert the tree, and return *its root*.

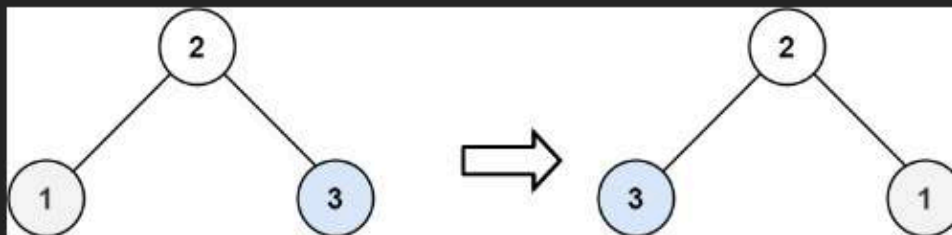
Example 1:



Input: `root = [4,2,7,1,3,6,9]`

Output: `[4,7,2,9,6,3,1]`

Example 2:



Input: `root = [2,1,3]`

Output: `[2,3,1]`

Example 3:

Input: `root = []`

Output: `[]`

</> Code

C ▾ 🔒 Auto

```
6  *    struct TreeNode *right;
7  * };
8  */
9  struct TreeNode* invertTree(struct TreeNode* root) {
10     if(root==NULL)
11         return NULL;
12     invertTree(root->left); //Call the left subtree
13     invertTree(root->right); //Call the right subtree
14     // Swap the nodes
15     struct TreeNode* temp = root->left;
16     root->left = root->right;
17     root->right = temp;
18     return root; // Return the root
19 }
20
```

Saved to local

Testcase

> Test Result

Accepted Runtime: 6 ms

Case 1

Case 2

Case 3

Input

root =
[4,2,7,1,3,6,9]

Output

[4,7,2,9,6,3,1]

Expected

[4,7,2,9,6,3,1]

♥

Contribute a testcase

Data Structures using C(23CS3PCDST)

Problem List < > ↺

Description Editorial Solutions Submissions

← All Submissions

Accepted

user8631x submitted at Feb 19, 2024 12:26

Editorial Solution

Runtime
3 ms
Beats 53.08% of users with C

Memory
6.03 MB
Beats 55.13% of users with C

Runtime (ms)	Percentage of users beaten
1ms	~42%
2ms	~2%
3ms	53.08%

Code | C

```
/**  
 * Definition for a binary tree node.  
 * struct TreeNode {  
 *     int val;  
 *     struct TreeNode *left;  
 *     struct TreeNode *right;  
 * };  
 */  
View more
```

More challenges

- 2415. Reverse Odd Levels of Binary Tree

Write your notes here

Code:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
struct TreeNode* invertTree(struct TreeNode* root) {
    if(root==NULL)
        return NULL;
    invertTree(root->left); //Call the left subtree
    invertTree(root->right); //Call the right subtree
    // Swap the nodes
    struct TreeNode* temp = root->left;
    root->left = root->right;
    root->right = temp;
    return root; // Return the root
}
```

Lab Program-9

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
// Define the maximum number of vertices in the graph
```

```
#define MAX_VERTICES 100
```

```
// Adjacency matrix representation of the graph
```

```
int graph[MAX_VERTICES][MAX_VERTICES];
```

```
// Function to add an edge between two vertices in the graph
```

```
void addEdge(int from, int to) {
    graph[from][to] = 1;
    graph[to][from] = 1;
}
```

// Function to perform BFS traversal of the graph

```
void bfs(int start) {
    int visited[MAX_VERTICES];
    for (int i = 0; i < MAX_VERTICES; i++) visited[i] = 0;
    int queue[MAX_VERTICES], front = 0, rear = 0;
    queue[rear++] = start;
    visited[start] = 1;
    while (front != rear) {
        int current = queue[front++];
        printf("%d ", current);
        for (int i = 0; i < MAX_VERTICES; i++) {
            if (graph[current][i] && !visited[i]) {
                queue[rear++] = i;
                visited[i] = 1;
            }
        }
    }
}
```

// Function to perform DFS traversal of the graph

```
void dfs(int start, int visited[], int* count) {
    visited[start] = 1;
    (*count)++;
    for (int i = 0; i < MAX_VERTICES; i++) {
        if (graph[start][i] && !visited[i]) {
            dfs(i, visited, count);
        }
    }
}
```

```
}
```

```
// Function to check whether the graph is connected or not
```

```
int isConnected(int vertices) {
    int visited[MAX_VERTICES];
    for (int i = 0; i < MAX_VERTICES; i++) visited[i] = 0;
    int count = 0;
    dfs(0, visited, &count);
    for (int i = 0; i < vertices; i++) {
        if (!visited[i]) return 0;
    }
    return 1;
}
```

```
int main() {
    // Construct the graph
    int vertices = 7;
    addEdge(0, 1);
    addEdge(0, 2);
    addEdge(1, 3);
    addEdge(1, 4);
    addEdge(2, 5);
    addEdge(2, 6);

    // Traverse the graph using BFS method
    printf("BFS traversal: ");
    bfs(0);
    printf("\n");

    // Check whether the graph is connected or not using DFS method
    if (isConnected(vertices)) {
        printf("The graph is connected.\n");
    } else {
```



```

        printf("The graph is not connected.\n");
    }

    return 0;
}

```

Output:

The screenshot shows a C program in a code editor and its output in a terminal window. The code defines a graph with 7 vertices and 6 edges, performs a BFS traversal, and checks if the graph is connected using DFS. The output shows the BFS traversal sequence and a message indicating the graph is connected.

```

main.c
67 // Construct the graph
68 int vertices = 7;
69 addEdge(0, 1);
70 addEdge(0, 2);
71 addEdge(1, 3);
72 addEdge(1, 4);
73 addEdge(2, 5);
74 addEdge(2, 6);
75
76 // Traverse the graph using BFS method
77 printf("BFS traversal: ");
78 bfs(0);
79 printf("\n");
80
81 // Check whether the graph is connected or not using DFS method
82 if (isConnected(vertices)) {
83     printf("The graph is connected.\n");
84 } else {
85     printf("The graph is not connected.\n");
86 }
87
88 return 0;
89 }

```

input

```

BFS traversal: 0 1 2 3 4 5 6
The graph is connected.

...Program finished with exit code 0
Press ENTER to exit console.

```

Lab Program-9b



HackerRank

Prepare > Data Structures > Linked Lists > Find Merge Point of Two Lists

Problem

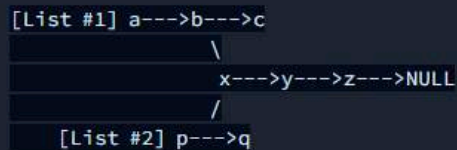
This challenge is part of a tutorial track by [MyCodeSchool](#)

Given pointers to the head nodes of 2 linked lists that merge together at some point, find the node where the two lists merge. The merge point is where both lists point to the same node, i.e. they reference the same memory location. It is guaranteed that the two head nodes will be different, and neither will be NULL. If the lists share a common node, return that node's *data* value.

Note: After the merge point, both lists will share the same node pointers.

Example

In the diagram below, the two lists converge at Node x:



Submissions

Leaderboard

Discussions

Editorial

Function Description

Complete the `findMergeNode` function in the editor below.

`findMergeNode` has the following parameters:

- `SinglyLinkedListNode` pointer `head1`: a reference to the head of the first list
- `SinglyLinkedListNode` pointer `head2`: a reference to the head of the second list

Returns

- `int`: the *data* value of the node where the lists merge

Input Format

Do not read any input from `stdin/console`.

The first line contains an integer t , the number of test cases.

Each of the test cases is in the following format:

The first line contains an integer, *index*, the node number where the merge will occur.

The next line contains an integer, *list1_{count}* that is the number of nodes in the first list.

Each of the following *list1_{count}* lines contains a *data* value for a node. The next line contains an integer, *list2_{count}* that is the number of nodes in the second list.

```
Change Theme Language C

> #include <assert.h>...

// Complete the findMergeNode function below.

/*
 * For your reference:
 *
 * SinglyLinkedListNode {
 *     int data;
 *     SinglyLinkedListNode* next;
 * };
 */
int findMergeNode(SinglyLinkedListNode* head1, SinglyLinkedListNode* head2) {
    while(head1){
        SinglyLinkedListNode *tmp = head1->next;
        head1->next = NULL;
        head1 = tmp;

        while(head2){
            if(head2->next == NULL){
                return head2->data;
            }
            head2 = head2->next;
        }
        return 0;
    }
}

94

> int main() ...
```

Data Structures using C(23CS3PCDST)

Line: 94 Col: 2

[Upload Code as File](#) ☐ Test against custom input [Run Code](#) [Submit Code](#)

Congratulations!

You have passed the sample test cases. Click the submit button to run your code against all the test cases.

✓ Sample Test case 0

✓ Sample Test case 1

Input (stdin)

1	1
2	1
3	3
4	1
5	2
6	3
7	1
8	1


Download

Your Output (stdout)

1	2
---	---

Data Structures using C(23CS3PCDST)

[Upload Code as File](#) ☐ Test against custom input [Run Code](#) [Submit Code](#)

 **You have earned 5.00 points!** 17% 5/3
You are now 25 points away from the 1st star for your problem solving badge.


Congratulations


You solved this challenge. Would you like to challenge your friends? [f](#) [t](#) [in](#)


[Next Challenge](#)


✓ Test case 0


✓ Test case 1

✓ Test case 2 

✓ Test case 3 

✓ Test case 4 

✓ Test case 5 

✓ Test case 6 

Input (stdin) [Download](#)

1	1
2	1
3	3
4	1
5	2
6	3
7	1
8	1

Expected Output [Download](#)

1	2
---	---