VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

Bhuvan. A (1BM24CS403)

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
September 2024-January 2025

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

This is to certify that the Lab work entitled **"DATA STRUCTURES"** carried out by **Bhuvan. A(1BM24CS403)**, 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 year 2024-25. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - **(23CS3PCDST)**work prescribed for the said degree.

Prof. Namratha M
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Kavitha Sooda
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Course outcomes:

CO1	Apply the concept of linear and nonlinear data structures.
CO2	Analyze data structure operations for a given problem
CO3	Design and develop solutions using the operations of linear and nonlinear data structure for a given specification.
CO4	Conduct practical experiments for demonstrating the operations of different data structures.

Lab program 1:

Write a program to simulate the working of stack using an array with the following:
a) Push
b) Pop
c) Display
The program should print appropriate messages for stack overflow, stack underflow.

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

void push(int *stack, int *top, int *max)
{
   int value;
   if(*top == *max-1){
      printf("Stack Overflow \n");
   }
   else{
      for (int i=0;i<*max;i++){
         printf("Enter value %d :\n",(i+1));
         scanf("%d",&value);
         stack[++*top]=value;
      }
   }
}
int pop(int *stack, int *top, int *max)
{
   if(*top== -1){
      printf("Stack Underflow \n");
      return -1;
   }
   else {
      return stack[(*top)--];
   }
}
void display(int *stack, int *top, int *max)
{
    if(*top== -1){
      printf("Stack Underflow \n");
      return -1;
   }
   else{
      for (int i=*top;i>=0;i--){
         printf("Value %d : %d \n",i+1,stack[i]);
      }
   }
}

void main()
{
   int max = 5;
```

```
    int stack[max];
    int top = -1;
    int choice;
    while(1){
        printf("Select among stack operations below \n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display\n");
        scanf("%d",&choice);
        switch(choice)
        {
        case 1:
            push(stack,&top,&max);
            break;
        case 2:
            printf("Successfully popped %d \n",pop(stack,&top,&max));
            break;
        case 3:
            display(stack,&top,&max);
            break;
        default:
            printf("invalid input ! \n");
            break;

        }
    }
}
```

Output:

```
Select among stack operations below
1. Push
2. Pop
3. Display
1
Enter value 1 :
10
Enter value 2 :
20
Enter value 3 :
30
Enter value 4 :
40
Enter value 5 :
50
Select among stack operations below
1. Push
2. Pop
3. Display
1
Stack Overflow
Select among stack operations below
1. Push
2. Pop
3. Display
2
Successfully popped 50
Select among stack operations below
1. Push
2. Pop
3. Display
2
Successfully popped 40
Select among stack operations below
1. Push
2. Pop
3. Display
3
Value 3 : 30
Value 2 : 20
Value 1 : 10
Select among stack operations below
1. Push
2. Pop
3. Display
```

```
1. Push
2. Pop
3. Display

Enter your choice :2

45 item was deleted
1. Push
2. Pop
3. Display

Enter your choice :2

65 item was deleted
1. Push
2. Pop
3. Display

Enter your choice :3
12
1. Push
2. Pop
3. Display

Enter your choice :2

12 item was deleted
1. Push
2. Pop
3. Display

Enter your choice :2
Stack underflow

1. Push
2. Pop
3. Display

Enter your choice :4

Invalid choice!!!
```

Lab Program 2:

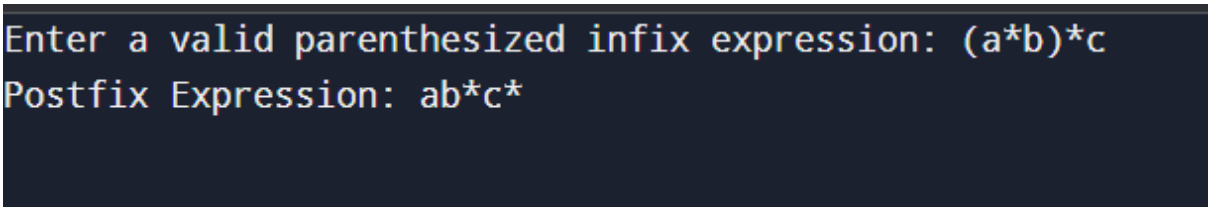
WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide)

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#define MAX 100
typedef struct {
    char data[MAX];
    int top;
} Stack;
void push(Stack *stack, char c) {
    if (stack->top == MAX - 1) {
        printf("Stack overflow\n");
        exit(1);
    }
    stack->data[++stack->top] = c;
}
char pop(Stack *stack) {
    if (stack->top == -1) {
        printf("Stack underflow\n");
        exit(1);
    }
    return stack->data[stack->top--];
}
char peek(Stack *stack) {
    if (stack->top == -1) {
        return '\0';
```

```
    }
    return stack->data[stack->top];
}

int precedence(char op) {
    if (op == '+' || op == '-') return 1;
    if (op == '*' || op == '/') return 2;
    return 0;
}
int is_operator(char c) {
    return c == '+' || c == '-' || c == '*' || c == '/';
}
void infix_to_postfix(const char *infix, char *postfix) {
    Stack stack = { .top = -1 };
    int i = 0, j = 0;
    while (infix[i] != '\0') {
        if (isalnum(infix[i])) {
            postfix[j++] = infix[i];
        } else if (infix[i] == '(') {
            push(&stack, infix[i]);
        } else if (infix[i] == ')') {
            while (peek(&stack) != '(') {
                postfix[j++] = pop(&stack);
            }
            pop(&stack); // Remove '('
        } else if (is_operator(infix[i])) {
            while (stack.top != -1 && precedence(peek(&stack)) >= precedence(infix[i])) {
                postfix[j++] = pop(&stack);
            }
```

```
            push(&stack, infix[i]);
        }
        i++;
    }
    while (stack.top != -1) {
        postfix[j++] = pop(&stack);
    }
    postfix[j] = '\0';
}
int main() {
    char infix[MAX], postfix[MAX];
    printf("Enter a valid parenthesized infix expression: ");
    scanf("%s", infix);
    infix_to_postfix(infix, postfix);
    printf("Postfix Expression: %s\n", postfix);
   return 0;
}
```

Output:



Lab Program 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

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
typedef struct {
    int data[MAX];
    int front;
    int rear;
} Queue;
void initialize(Queue *q) {
    q->front = -1;
    q->rear = -1;
}
int is_empty(Queue *q) {
    return q->front == -1;
}
int is_full(Queue *q) {
    return q->rear == MAX - 1;
}
void insert(Queue *q, int value) {
    if (is_full(q)) {
        printf("Queue Overflow: Cannot insert %d\n", value);
        return;
    }
    if (q->front == -1) {
        q->front = 0;
```

```
    }
    q->rear++;
    q->data[q->rear] = value;
    printf("Inserted %d into the queue.\n", value);
}
void delete(Queue *q) {
    if (is_empty(q)) {
        printf("Queue Underflow: Cannot delete from an empty queue.\n");
        return;
    }
    printf("Deleted %d from the queue.\n", q->data[q->front]);
    if (q->front == q->rear) {
        q->front = -1;
        q->rear = -1;
    } else {
        q->front++;
    }
}
void display(Queue *q) {
    if (is_empty(q)) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Queue contents: ");
    for (int i = q->front; i <= q->rear; i++) {
        printf("%d ", q->data[i]);
    }
    printf("\n");
}
```

```
int main() {
    Queue q;
    int choice, value;
    initialize(&q);
    while (1) {
        printf("\nQueue Operations:\n");
        printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter the value to insert: ");
                scanf("%d", &value);
                insert(&q, value);
                break;
            case 2:
                delete(&q);
                break;
            case 3:
                display(&q);
                break;
            case 4:
                printf("Exiting the program.\n");
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
```

```
    return 0;
}
```

Output:

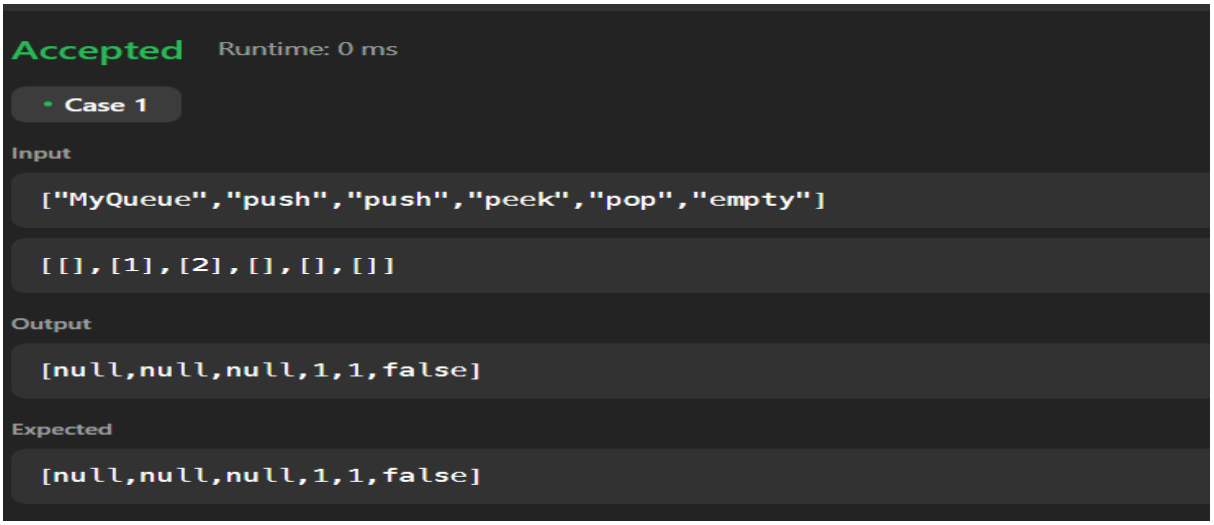
```
Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 10
Inserted 10 into the queue.

Enter your choice: 1
Enter the value to insert: 20
Inserted 20 into the queue.

Enter your choice: 3
Queue contents: 10 20

Enter your choice: 2
Deleted 10 from the queue.

Enter your choice: 3
Queue contents: 20

Enter your choice: 4
Exiting the program.
```

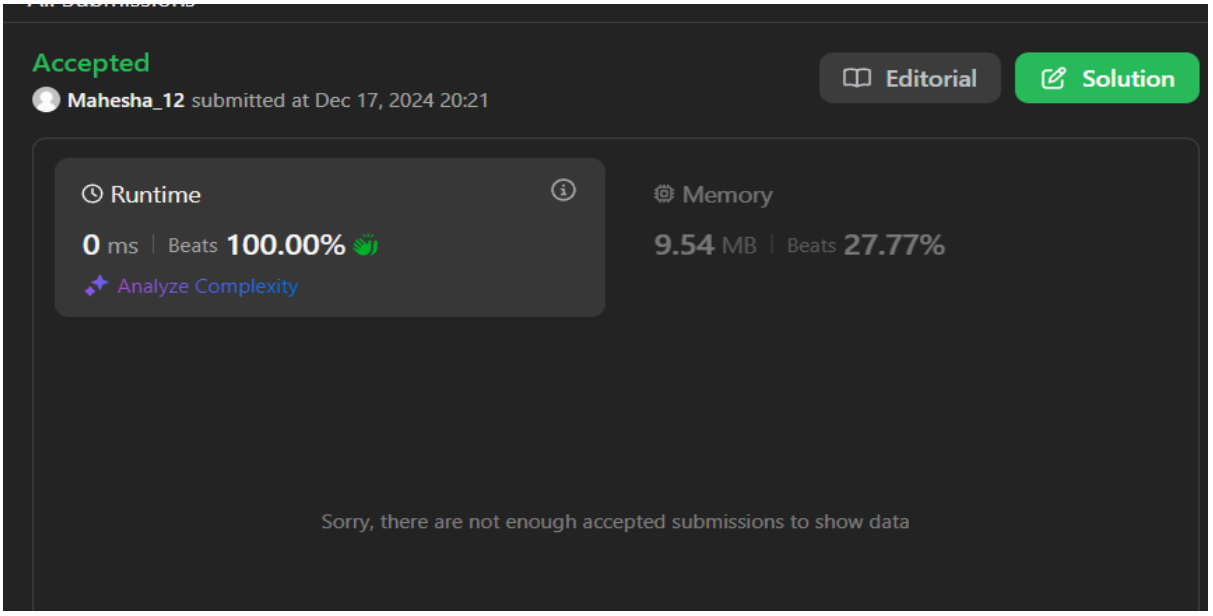
LeetCode 1:

Implement Queue using Stacks

```
class MyQueue {
public:
        stack<int> s1,s2;
        MyQueue() {

        }
        void push(int x) {
                while(!s1.empty())
```

```
            {
                s2.push(s1.top());
                s1.pop();
            }
            s1.push(x);
            while(!s2.empty())
                s1.push(s2.top());
                s2.pop();
            }
        }
        int pop() {
            if(s1.empty())  return -1;
            int ans = s1.top();
            s1.pop();
            return ans;
        }

        int peek() {
            return s1.empty()?-1:s1.top();
        }

        bool empty() {
            return s1.empty();
        }
};
```


Accepted
Runtime: 0 ms
Case 1
Input
["MyQueue","push","push","peek","pop","empty"]
[[],[1],[2],[],[],[]]
Output
[null,null,null,1,1,false]
Expected
[null,null,null,1,1,false]


Accepted
Mahesha_12 submitted at Dec 17, 2024 20:21
Editorial
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
9.54 MB | Beats 27.77%
Sorry, there are not enough accepted submissions to show data

3B(I) 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. The program should be done using pass by reference only.

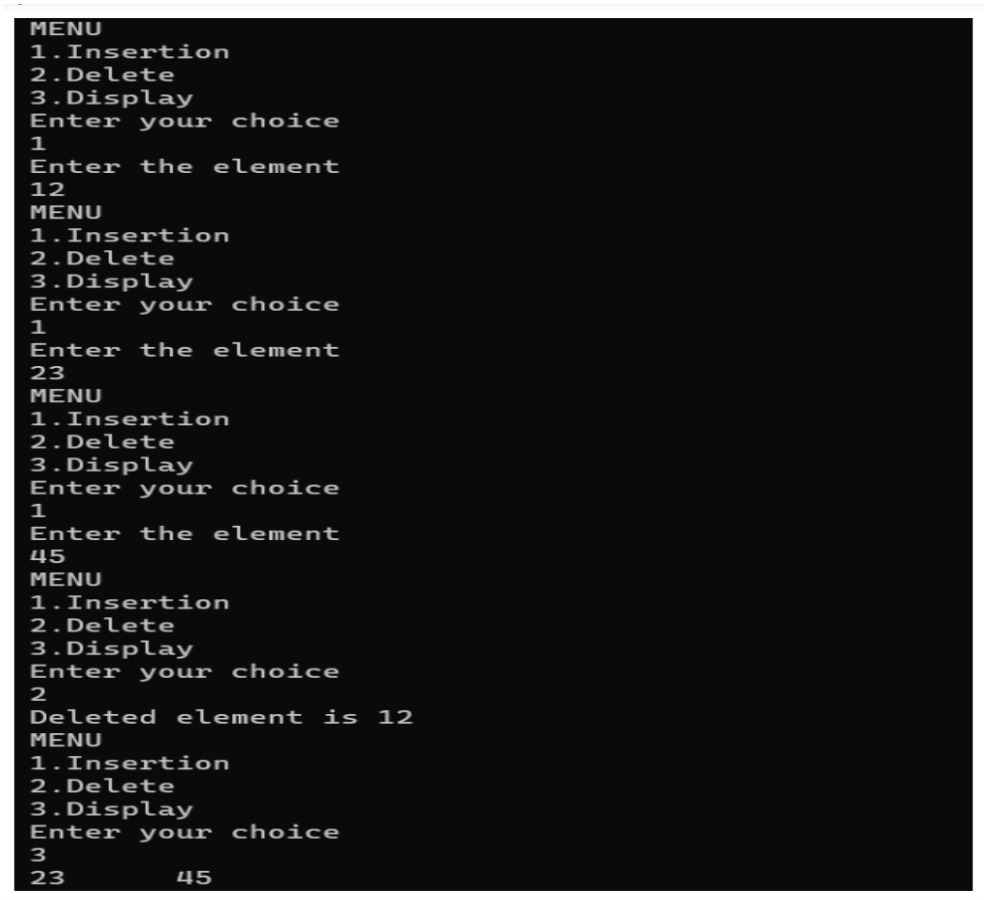
```
#include <stdio.h>

#include <stdlib.h>

#define MAX 5

typedef struct {

int data[MAX];

int front;

int rear;

} CircularQueue;

void initializeQueue(CircularQueue *q) {

q->front = -1;

q->rear = -1;

}

int isFull(CircularQueue *q) {

return (q->rear + 1) % MAX == q->front;

}


int isEmpty(CircularQueue *q) {

return q->front == -1;

}


void insert(CircularQueue *q, int value) {

if (isFull(q)) {

printf("Queue Overflow! Cannot insert %d\n", value);

return;

}

if (isEmpty(q)) {

q->front = 0;
```

```
}
q->rear = (q->rear + 1) % MAX;
q->data[q->rear] = value;
printf("Inserted %d into the queue\n", value);
}

int delete(CircularQueue *q) {
if (isEmpty(q)) {
printf("Queue Underflow! Cannot delete\n");
return -1;
}
int value = q->data[q->front];
if (q->front == q->rear) {
q->front = -1;
q->rear = -1;
} else {
q->front = (q->front + 1) % MAX;
}
printf("Deleted %d from the queue\n", value);
return value;
}

void display(CircularQueue *q) {
if (isEmpty(q)) {
printf("Queue is empty\n");
return;
}
printf("Queue elements: ");
int i = q->front;
```

```
while (1) {
printf("%d ", q->data[i]);
if (i == q->rear) break;
i = (i + 1) % MAX;
}
printf("\n");
}

int main() {
CircularQueue q;
initializeQueue(&q);
int choice, value;

while (1) {
printf("\n1. Insert \n2. Delete \n3. Display \n4. Exit");
printf("\nChoose an operation: ");
scanf("%d", &choice);

switch (choice) {
        case 1:
            printf("Enter value to insert: ");
            scanf("%d", &value);
            insert(&q, value);
            break;
        case 2:
            delete(&q);
            break;
        case 3:
            display(&q);
```

```
            break;
        case 4:
            printf("Exiting...\n");
            return 0;
        default:
            printf("Invalid choice, please try again.\n");
    }
}
```

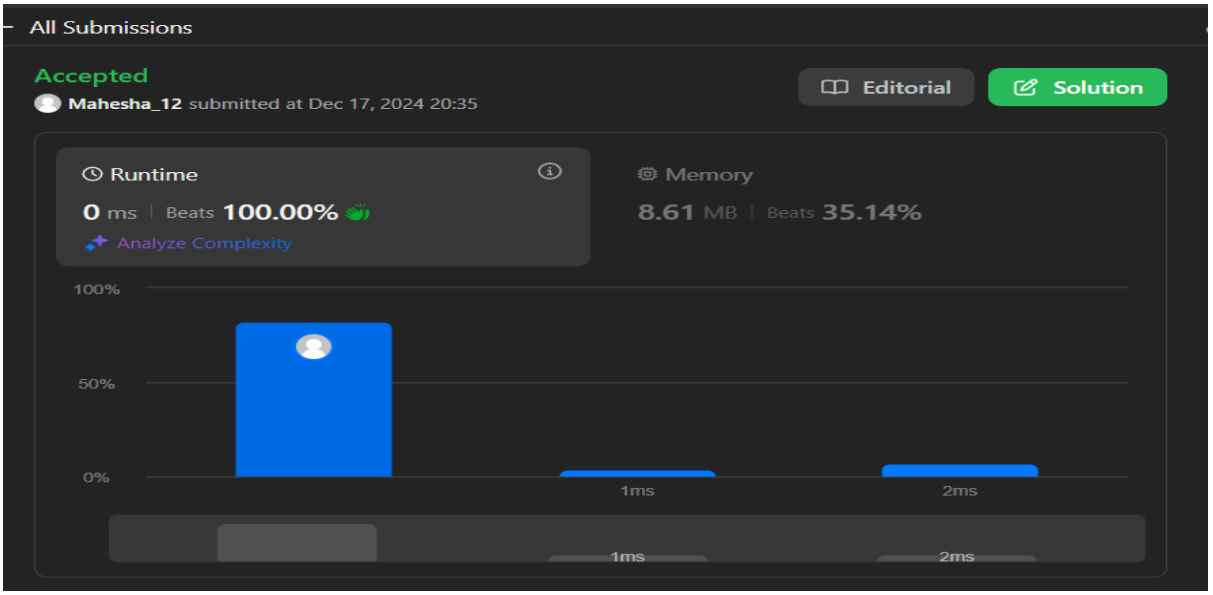
OUTPUT:



(II) Valid Parentheses

```
class Solution {
public:
    bool isValid(string s) {
        stack<char> st ;
        for (int i = 0 ;  i< s.length() ; i++)
        {
            char ch = s[i];
            // if opening bracket then push into the stack
            if (ch == '(' || ch == '{' || ch == '[')
            {
                st.push(ch) ;
            }
            else {
                // if a closing bracket then we compare with the top of the stack
                // while comparing with top of stack we have 2 cases
                // the stack can be empty or the stack is not empty
                if (!st.empty())
                {
                    char top = st.top() ;
                    if ((ch == ')' && top == '(') ||
                        (ch == '}' && top == '{') ||
                        (ch == ']' && top == '['))
                        {
                            // if matches then pop
                            st.pop() ;
                        }
                        else
                        {
```

```
                return false ;
            }
        }
        else
        {
            // if stack is empty and we get a closing bracket means the string is unbalanced
            return false ;
        }
      }
    }

    // in the end if the stack is empty -- meaning there is no opening bracket present in the stack -- meaning all opening brackets have found their corresponding closing bracket and have been popped then we return trie
    if (st.empty())
    {
       return true ;
    }
    return false ;
  }
};
```


All Submissions
Accepted
Mahesha_12 submitted at Dec 17, 2024 20:35
Editorial
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
8.61 MB | Beats 35.14%
100%
50%
0%
1ms
2ms
1ms
2ms


Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Case 4
Input
s =
"()"
Output
true
Expected
true

LAB PROGRAM 4

WAP to Implement Singly Linked List with following operations a) Create a linked list. b) Insertion of a node at first position, at any position and at end of list. c) Display the contents of the linked list.

```
#include <stdio.h>

#include <stdlib.h>



struct Node {

    int data;

    struct Node* next;

};


struct Node* createNode(int data) {

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

    newNode->data = data;

    newNode->next = NULL;

    return newNode;

}


void insertAtFirst(struct Node** head, int data) {

    struct Node* newNode = createNode(data);

    newNode->next = *head;

    *head = newNode;

}


void insertAtEnd(struct Node** head, int data) {

    struct Node* newNode = createNode(data);

    if (*head == NULL) {

        *head = newNode;
```

```
        return;
    }
    struct Node* last = *head;
    while (last->next != NULL) {
        last = last->next;
    }
    last->next = newNode;
}

void insertAtPosition(struct Node** head, int data, int position) {
    if (position == 0) {
        insertAtFirst(head, data);
        return;
    }
    struct Node* newNode = createNode(data);
    struct Node* current = *head;
    for (int i = 0; current != NULL && i < position - 1; i++) {
        current = current->next;
    }
    if (current == NULL) {
        printf("Position out of bounds.\n");
        free(newNode);
        return;
    }
    newNode->next = current->next;
    current->next = newNode;
}
```

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

int main() {
    struct Node* head = NULL;
     int choice, data, position;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Insert at First\n");
        printf("2. Insert at End\n");
        printf("3. Insert at Position\n");
        printf("4. Delete First Element\n");
        printf("5. Delete Last Element\n");
        printf("6. Delete Specified Element\n");
        printf("7. Display List\n");
        printf("8. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
```

```
            printf("Enter data to insert at first: ");
            scanf("%d", &data);
            insertAtFirst(&head, data);
            break;
        case 2:
            printf("Enter data to insert at end: ");
            scanf("%d", &data);
            insertAtEnd(&head, data);
            break;
        case 3:
            printf("Enter data to insert at position: ");
            scanf("%d", &data);
            printf("Enter position: ");
            scanf("%d", &position);
            insertAtPosition(&head, data, position);
            break;
        case 4:
            display(head);
            break;
        case 5:
            exit(0);
        default:
            printf("Invalid choice. Please try again.\n");
    }
  }

  return 0;
}
```

```
--- Insertion Menu ---
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
Enter your choice: 1
Enter value to insert: 12

--- Insertion Menu ---
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
Enter your choice: 1
Enter value to insert: 14

--- Insertion Menu ---
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
Enter your choice: 2
Enter value to insert: 14

--- Insertion Menu ---
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
Enter your choice: 2
Enter value to insert: 5
```

```
--- Insertion Menu ---
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
Enter your choice: 2
Enter value to insert: 5

--- Insertion Menu ---
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
Enter your choice: 3
Enter value and position: 3
3

--- Insertion Menu ---
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
Enter your choice: 4
Linked List: 14 -> 12 -> 3 -> 14 -> 5 -> NULL
```

LAB PROGRAM 5

WAP to Implement Singly Linked List with following operations

a) Create a linked list.

b) Deletion of first element, specified element and last element in the list.

c) Display the contents of the linked list.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
void deleteFirst(struct Node** head) {
    if (*head == NULL) return;
    struct Node* temp = *head;
    *head = (*head)->next;
    free(temp);
}

void deleteLast(struct Node** head) {
    if (*head == NULL) return;
```

```
    if ((*head)->next == NULL) {
        free(*head);
        *head = NULL;
        return;
    }
    struct Node* secondLast = *head;
    while (secondLast->next->next != NULL) {
        secondLast = secondLast->next;
    }
    free(secondLast->next);
    secondLast->next = NULL;
}

void deleteElement(struct Node** head, int data) {
    if (*head == NULL) return;
    if ((*head)->data == data) {
        deleteFirst(head);
        return;
    }
    struct Node* current = *head;
    while (current->next != NULL && current->next->data != data) {
        current = current->next;
    }
    if (current->next == NULL) return;
    struct Node* temp = current->next;
    current->next = current->next->next;
    free(temp);
}
```

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

int main() {
    struct Node* head = NULL;
     int choice, data, position;

    while (1) {
        printf("\nMenu:\n");
        printf("4. Delete First Element\n");
        printf("5. Delete Last Element\n");
        printf("6. Delete Specified Element\n");
        printf("7. Display List\n");
        printf("8. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 4:
                deleteFirst(&head);
                printf("First element deleted.\n");
                break;
            case 5:
```

```
            deleteLast(&head);
            printf("Last element deleted.\n");
            break;
        case 6:
            printf("Enter data to delete: ");
            scanf("%d", &data);
            deleteElement(&head, data);
            printf("Element deleted.\n");
            break;
        case 7:
            display(head);
            break;
        case 8:
            exit(0);
        default:
            printf("Invalid choice. Please try again.\n");
      }
   }

   return 0;
}
```

```
--- Deletion Menu ---
1. Insert at beginning
2. Delete at beginning
3. Delete at end
4. Delete a value
5. Display
6. Exit
Enter your choice: 1
Enter value to insert: 23

--- Deletion Menu ---
1. Insert at beginning
2. Delete at beginning
3. Delete at end
4. Delete a value
5. Display
6. Exit
Enter your choice: 5
Linked List: 23 -> 78 -> 45 -> 51 -> 4 -> NULL

--- Deletion Menu ---
1. Insert at beginning
2. Delete at beginning
3. Delete at end
4. Delete a value
5. Display
6. Exit
Enter your choice: 2
Deleted value: 23

--- Deletion Menu ---
1. Insert at beginning
2. Delete at beginning
3. Delete at end
4. Delete a value
5. Display
6. Exit
Enter your choice: 3
Deleted value: 4
```

```
--- Deletion Menu ---
1. Insert at beginning
2. Delete at beginning
3. Delete at end
4. Delete a value
5. Display
6. Exit
Enter your choice: 4
Enter value to delete: 45
Deleted value: 45

--- Deletion Menu ---
1. Insert at beginning
2. Delete at beginning
3. Delete at end
4. Delete a value
5. Display
6. Exit
Enter your choice: 5
Linked List: 78 -> 51 -> NULL
```

Lab Program 6:

A. WAP to Implement Single Link List with following operations: Sort the linked list,

```
#include <stdio.h>
#include <stdlib.h>
struct node {
 int value;
 struct node *next;
};
typedef struct node *NODE;
NODE getnode() {
 NODE ptr;
 ptr = (NODE)malloc(sizeof(struct node));
 if (ptr == NULL) {
 printf("Memory is not allocated");
 return NULL;
 }
 return ptr;
}
NODE insert_beg(NODE first, int item) {
 NODE new;
 new = getnode();
 new->value = item;
 new->next = NULL;
 if (first == NULL) {
 return new;
 }
 new->next = first;
```

```
 return new;
31 | P a g e
}
void display(NODE first) {
 NODE temp;
 if (first == NULL) {
 printf("Linked list is empty\n");
 return;
 }
 temp = first;
 while (temp != NULL) {
 printf("%d \t", temp->value);
 temp = temp->next;
 }
 printf("NULL\n");
}
NODE concatinate(NODE first1,NODE first2)
{
 if(first1==NULL&&first2==NULL)
 return NULL;
 if(first1==NULL)
 return first2;
 if(first2==NULL)
 return first1;
 NODE last;
 last=first1;
 while(last->next!=NULL)
```

```
 last=last->next;
 last->next=first2;
 return first1;
}
NODE reverse(NODE first)
{
```

```
 NODE current, temp;
 current=NULL;
 while(first!=NULL)
 {
 temp=first;
 first=first->next;
 temp->next=current;
 current=temp;
 }
 return current;
}
void sort(NODE first)
{
 int x;
 NODE temp1,temp2;
 temp1=first;
 temp2=first->next;
 while(temp1->next!=NULL)
 {
 while(temp2!=NULL)
```

```
{
if((temp1->value)>=(temp2->value))
{
x=temp1->value;
temp1->value=temp2->value;
temp2->value=x;
}
temp2=temp2->next;
}
temp1=temp1->next;
}
}
```



```
int main() {
 NODE first1,first2;
 first1 = NULL;
 first2=NULL;
 int choice, val;
 while (1) {
 printf("---MENU---\n");
 printf("1. Insertion in Linked list 1\n");
 printf("2. Insertion in Linked list 2\n");
 printf("3. Concatination\n");
 printf("4. Reverse\n");
 printf("5. Sort\n");
 printf("6. Display linked list 1\n");
 printf("7. Display linked list 2\n");
```

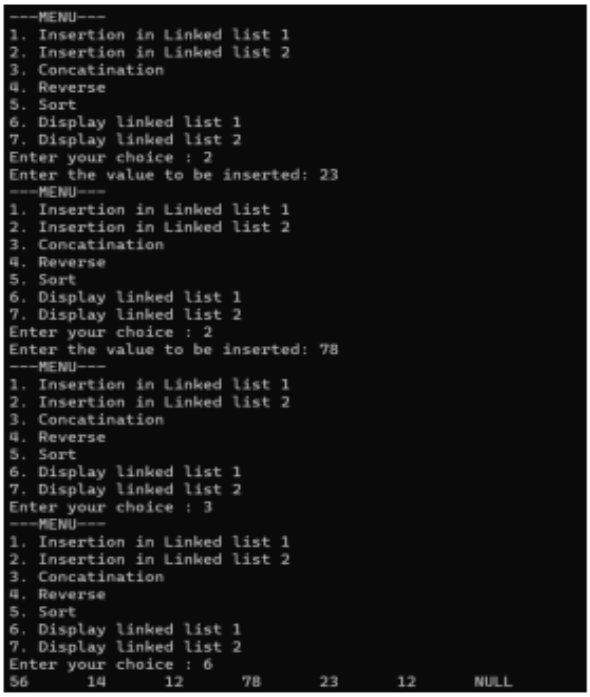
```
printf("Enter your choice : ");
scanf("%d",&choice);
switch(choice)
{
case 1:
printf("Enter the value to be inserted: ");
scanf("%d", &val);
first1 = insert_beg(first1, val);
break;
case 2:
printf("Enter the value to be inserted: ");
scanf("%d", &val);
first2 = insert_beg(first2, val);
break;
case 3:
first1=concatinate(first1,first2);
break;
```

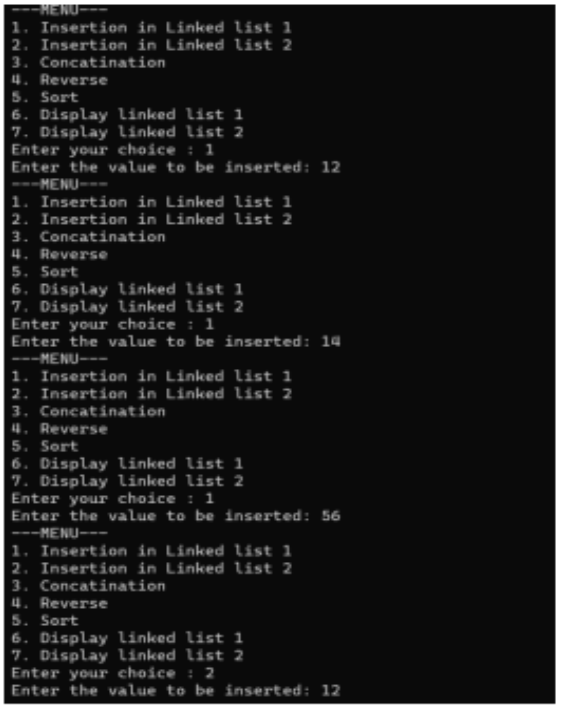


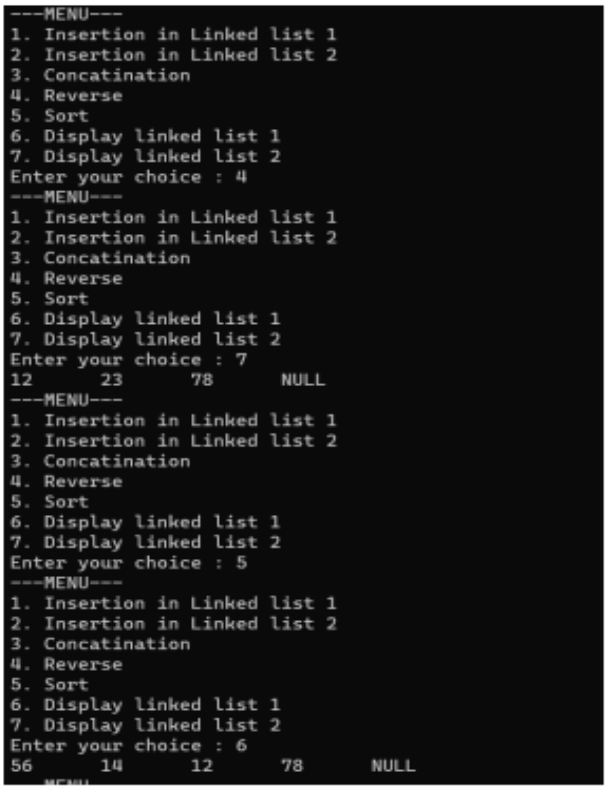
```
case 4:
first2=reverse(first2);
break;
case 5:
sort(first2);
break;
case 6:
display(first1);
break;
```

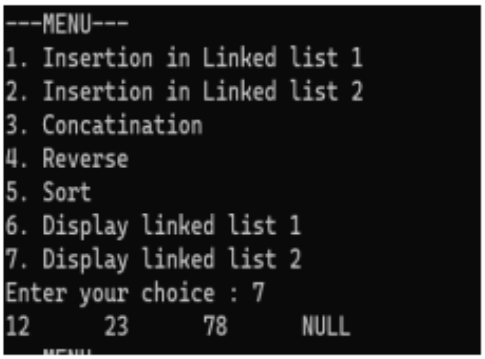
```
case 7:
display(first2);
break;
default:
printf("Enter a valid choice\n");
break;
}
}
return 0;
}
```
reverse the linked list, Concatenation of two linked lists.

OUTPUT:


---MENU---
1. Insertion in Linked list 1
2. Insertion in Linked list 2
3. Concatination
4. Reverse
5. Sort
6. Display linked list 1
7. Display linked list 2
Enter your choice : 2
Enter the value to be inserted: 23
---MENU---
1. Insertion in Linked list 1
2. Insertion in Linked list 2
3. Concatination
4. Reverse
5. Sort
6. Display linked list 1
7. Display linked list 2
Enter your choice : 2
Enter the value to be inserted: 78
---MENU---
1. Insertion in Linked list 1
2. Insertion in Linked list 2
3. Concatination
4. Reverse
5. Sort
6. Display linked list 1
7. Display linked list 2
Enter your choice : 3
---MENU---
1. Insertion in Linked list 1
2. Insertion in Linked list 2
3. Concatination
4. Reverse
5. Sort
6. Display linked list 1
7. Display linked list 2
Enter your choice : 6
56 14 12 78 23 12 NULL


---MENU---
1. Insertion in Linked list 1
2. Insertion in Linked list 2
3. Concatination
4. Reverse
5. Sort
6. Display linked list 1
7. Display linked list 2
Enter your choice : 1
Enter the value to be inserted: 12
---MENU---
1. Insertion in Linked list 1
2. Insertion in Linked list 2
3. Concatination
4. Reverse
5. Sort
6. Display linked list 1
7. Display linked list 2
Enter your choice : 1
Enter the value to be inserted: 14
---MENU---
1. Insertion in Linked list 1
2. Insertion in Linked list 2
3. Concatination
4. Reverse
5. Sort
6. Display linked list 1
7. Display linked list 2
Enter your choice : 1
Enter the value to be inserted: 56
---MENU---
1. Insertion in Linked list 1
2. Insertion in Linked list 2
3. Concatination
4. Reverse
5. Sort
6. Display linked list 1
7. Display linked list 2
Enter your choice : 2
Enter the value to be inserted: 12


---MENU---
1. Insertion in Linked list 1
2. Insertion in Linked list 2
3. Concatination
4. Reverse
5. Sort
6. Display linked list 1
7. Display linked list 2
Enter your choice : 4
---MENU---
1. Insertion in Linked list 1
2. Insertion in Linked list 2
3. Concatination
4. Reverse
5. Sort
6. Display linked list 1
7. Display linked list 2
Enter your choice : 7
12 23 78 NULL
---MENU---
1. Insertion in Linked list 1
2. Insertion in Linked list 2
3. Concatination
4. Reverse
5. Sort
6. Display linked list 1
7. Display linked list 2
Enter your choice : 5
---MENU---
1. Insertion in Linked list 1
2. Insertion in Linked list 2
3. Concatination
4. Reverse
5. Sort
6. Display linked list 1
7. Display linked list 2
Enter your choice : 6
56 14 12 78 NULL


---MENU---
1. Insertion in Linked list 1
2. Insertion in Linked list 2
3. Concatination
4. Reverse
5. Sort
6. Display linked list 1
7. Display linked list 2
Enter your choice : 7
12 23 78 NULL

B] WAP to Implement Single Link List to simulate Stack & Queue Operations.

```
#include <stdio.h>
#include <stdlib.h>
struct node {
 int value;
 struct node *next;
};
typedef struct node *NODE;
NODE getnode() {
 NODE ptr;
 ptr = (NODE)malloc(sizeof(struct node));
 if (ptr == NULL) {
 printf("Memory is not allocated");
 return NULL;
 }
 return ptr;
}
NODE insert_beg(NODE first, int item) {
 NODE new;
 new = getnode();
 new->value = item;
 new->next = NULL;
 if (first == NULL) {
 return new;
 }
 new->next = first;
```

```
 return new;
}
NODE del_beg(NODE first) {
 if (first == NULL) {
 printf("Linked list is empty\n");
 return NULL;
 }
 NODE temp;
 temp = first;
 first = temp->next;
 free(temp);
 return first;
}
void display(NODE first) {
 NODE temp;
 if (first == NULL) {
 printf("Linked list is empty\n");
 return;
 }
 temp = first;
 while (temp != NULL) {
 printf("%d\t", temp->value);
 temp = temp->next;
 }
 printf("NULL\n");
}
NODE insert_end(NODE first, int item) {
```

```
 NODE new_end, current;
 new_end = getnode();
 new_end->value = item;
 new_end->next = NULL;
 if (first == NULL) {
 return new_end;
}
 current = first;
 while (current->next != NULL) {
 current = current->next;
 }
 current->next = new_end;
 return first;
}
int main() {
 NODE first;
 first = NULL;
 int choice, val;
 do {
 printf("STACK IMPLEMENTATION\n");
 printf("1. Push\n");
 printf("2. Pop\n");
 printf("3. Display\n");
 printf("4.Exit\n");
 printf("Enter your choice: ");
 scanf("%d", &choice);
 switch (choice) {
```

```
 case 1:
 printf("Enter the value to be inserted: ");
 scanf("%d", &val);
 first = insert_beg(first, val);
 break;
 case 2:
 first = del_beg(first);
 break;
 case 3:
 display(first);
 break;
default:
 printf("Enter a valid choice\n");
 break;
 }}while( choice!=4);
 first=NULL;
 while (1) {
 printf("QUEUE IMPLEMENTATION\n");
 printf("1. Insert\n");
 printf("2. Delete\n");
 printf("3. Display\n");
 printf("4.Exit\n");
 printf("Enter your choice: ");
 scanf("%d", &choice);
 switch (choice) {
 case 1:
 printf("Enter the value to be inserted: ");
```

```
scanf("%d", &val);
first = insert_end(first, val);
break;
case 2:
first = del_beg(first);
break;
case 3:
display(first);
break;
case 4: exit(0);
default:
printf("Enter a valid choice\n");
break;
}
}
return 0;
}
```

Output:

```
STACK IMPLEMENTATION
1. Push
2. Pop
3. Display
4.Exit
Enter your choice: 1
Enter the value to be inserted: 12
STACK IMPLEMENTATION
1. Push
2. Pop
3. Display
4.Exit
Enter your choice: 1
Enter the value to be inserted: 45
STACK IMPLEMENTATION
1. Push
2. Pop
3. Display
4.Exit
Enter your choice: 1
Enter the value to be inserted: 78
STACK IMPLEMENTATION
1. Push
2. Pop
3. Display
4.Exit
Enter your choice: 2
STACK IMPLEMENTATION
1. Push
2. Pop
3. Display
4.Exit
Enter your choice: 3
45      12      NULL
```

```
QUEUE IMPLEMENTATION
1. Insert
2. Delete
3. Display
4.Exit
Enter your choice: 1
Enter the value to be inserted: 14
QUEUE IMPLEMENTATION
1. Insert
2. Delete
3. Display
4.Exit
Enter your choice: 1
Enter the value to be inserted: 78
QUEUE IMPLEMENTATION
1. Insert
2. Delete
3. Display
4.Exit
Enter your choice: 1
Enter the value to be inserted: 25
QUEUE IMPLEMENTATION
1. Insert
2. Delete
3. Display
4.Exit
Enter your choice: 2
QUEUE IMPLEMENTATION
1. Insert
2. Delete
3. Display
4.Exit
Enter your choice: 3
78      25      NULL
```

LAB PROGRAM 7

A] WAP to Implement doubly link list with primitive operations

a) Create a doubly linked list.

b) Insert a new node to the left of the node.

c) Delete the node based on a specific value

d) Display the contents of the list

```
#include <stdio.h>

#include <stdlib.h>


struct Node {

   int data;

   struct Node* next;

   struct Node* prev;

};


void insertAtBegin(struct Node** head, int value) {

   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

   newNode->data = value;

   newNode->next = *head;

   newNode->prev = NULL;


   if (*head != NULL) {

      (*head)->prev = newNode;

   }


   *head = newNode;

   printf("Node with value %d inserted at the beginning.\n", value);
```

```
}

void deleteAtLast(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty. Nothing to delete.\n");
        return;
    }

    struct Node* temp = *head;

    while (temp->next != NULL) {
        temp = temp->next;
    }

    if (temp->prev != NULL) {
        temp->prev->next = NULL;
    } else {
        *head = NULL;
    }

    free(temp);
    printf("Last node deleted.\n");
}

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

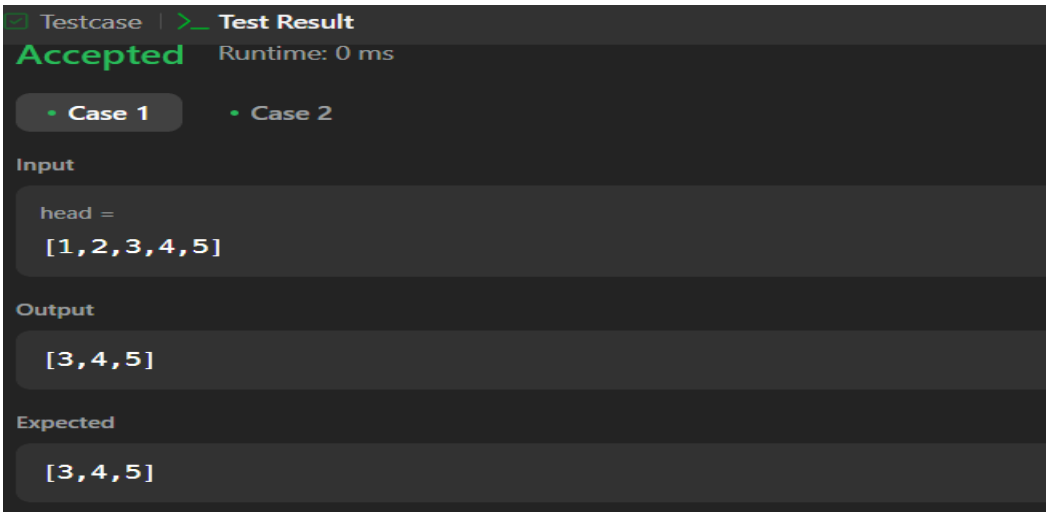
```
        return;
    }

    struct Node* temp = head;
    printf("Doubly Linked List: ");

    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL");
}

int main() {
    struct Node* head = NULL;
    int choice, value;

    do {
        printf("\nMenu:\n");
        printf("1. Insert node at the beginning\n");
        printf("2. Delete node at the end\n");
        printf("3. Print the list\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
```

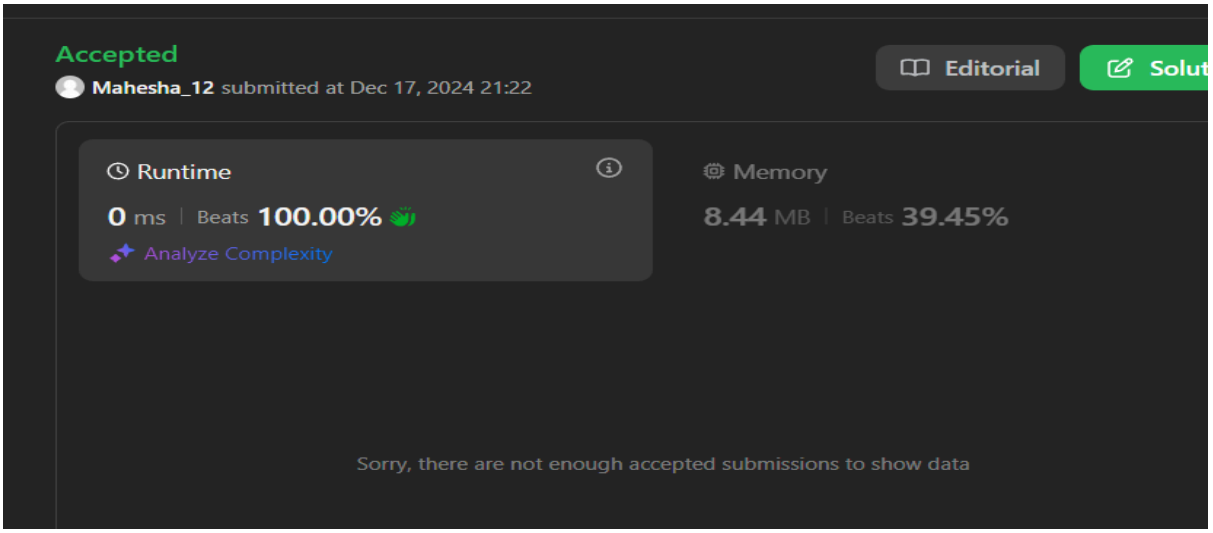
```
            case 1:
                printf("Enter value to insert at the beginning: ");
                scanf("%d", &value);
                insertAtBegin(&head, value);
                break;

            case 2:
                deleteAtLast(&head);
                break;

            case 3:
                printList(head);
                break;

            case 4:
                printf("Exiting program...\n");
                break;

            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 4);

    return 0;
}
```

```
Menu:
1. Insert node at the beginning
2. Delete node at the end
3. Print the list
4. Exit
Enter your choice: 1
Enter value to insert at the beginning: 30
Node with value 30 inserted at the beginning.

Menu:
1. Insert node at the beginning
2. Delete node at the end
3. Print the list
4. Exit
Enter your choice: 1
Enter value to insert at the beginning: 20
Node with value 20 inserted at the beginning.

Menu:
1. Insert node at the beginning
2. Delete node at the end
3. Print the list
4. Exit
Enter your choice: 3
Doubly Linked List: 20 -> 30 -> 20 -> NULL
Menu:
1. Insert node at the beginning
2. Delete node at the end
3. Print the list
4. Exit
Enter your choice: 2
Last node deleted.

Menu:
1. Insert node at the beginning
2. Delete node at the end
3. Print the list
4. Exit
Enter your choice: 3
Doubly Linked List: 20 -> 30 -> NULL
Menu:
1. Insert node at the beginning
2. Delete node at the end
3. Print the list
4. Exit
Enter your choice: 1
Enter value to insert at the beginning: 80
Node with value 80 inserted at the beginning.

Menu:
1. Insert node at the beginning
2. Delete node at the end
3. Print the list
4. Exit
Enter your choice: 3
Doubly Linked List: 80 -> 20 -> 30 -> NULL
Menu:
```

B] Leetcode Program:

```
struct ListNode* middleNode(struct ListNode* head) {
struct ListNode* one=head;
struct ListNode* two=head;
while(two!=NULL&& two->next!=NULL)
{
one=one->next;
two=two->next->next;
}
return one;
}
```





LAB PROGRAM 8

Write a program

a) To construct a binary Search tree.

b) To traverse the tree using all the methods i.e., in-order, preorder and post order

c) To display the elements in the tree.

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

struct Node {
    int key;
    struct Node* left;
    struct Node* right;
};

struct Node* createNode(int key) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->key = key;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

struct Node* insert(struct Node* root, int key) {
    if (root == NULL) {
        return createNode(key);
    }
    if (key < root->key) {
```

```
        root->left = insert(root->left, key);
    } else if (key > root->key) {
        root->right = insert(root->right, key);
    }
    return root;
}

void inorder(struct Node* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->key);
        inorder(root->right);
    }
}

void preorder(struct Node* root) {
    if (root != NULL) {
        printf("%d ", root->key);
        preorder(root->left);
        preorder(root->right);
    }
}

void postorder(struct Node* root) {
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
```

```
        printf("%d ", root->key);
    }
}

int main() {
    struct Node* root = NULL;
    int choice, value;

    do {
        printf("\n1. Insert\n2. In-order Traversal\n3. Pre-order Traversal\n4. Post-order Traversal\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                root = insert(root, value);
                break;
            case 2:
                printf("In-order traversal: ");
                inorder(root);
                printf("\n");
                break;
            case 3:
                printf("Pre-order traversal: ");
```

```
            preorder(root);
            printf("\n");
            break;
        case 4:
            printf("Post-order traversal: ");
            postorder(root);
            printf("\n");
            break;
        case 5:
            printf("Exiting program.\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
      }
   } while (choice != 5);

   return 0;
}
```

Output:

```
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter value to insert: 40

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter value to insert: 15

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 2
In-order traversal: 10 15 20 40 50

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 3
Pre-order traversal: 20 10 15 50 40

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 4
Post-order traversal: 15 10 40 50 20
```

B] LeetCode Program:

```
struct ListNode *getIntersectionNode(struct ListNode *headA, struct ListNode *headB) {
 if(headA==NULL||headB==NULL)
 return NULL;
 struct ListNode *ptr1=headA;
 struct ListNode *ptr2=headB;
 while(ptr1!=ptr2)
 {
 ptr1=(ptr1==NULL)?headB:ptr1->next;
 ptr2=(ptr2==NULL)?headA:ptr2->next;
 }
 return ptr1;
}
```

Testcase | Test Result

Accepted Runtime: 3 ms

• Case 1 • Case 2 • Case 3

Input

intersectVal =
8

listA =
[4,1,8,4,5]

listB =
[5,6,1,8,4,5]

skipA =
2

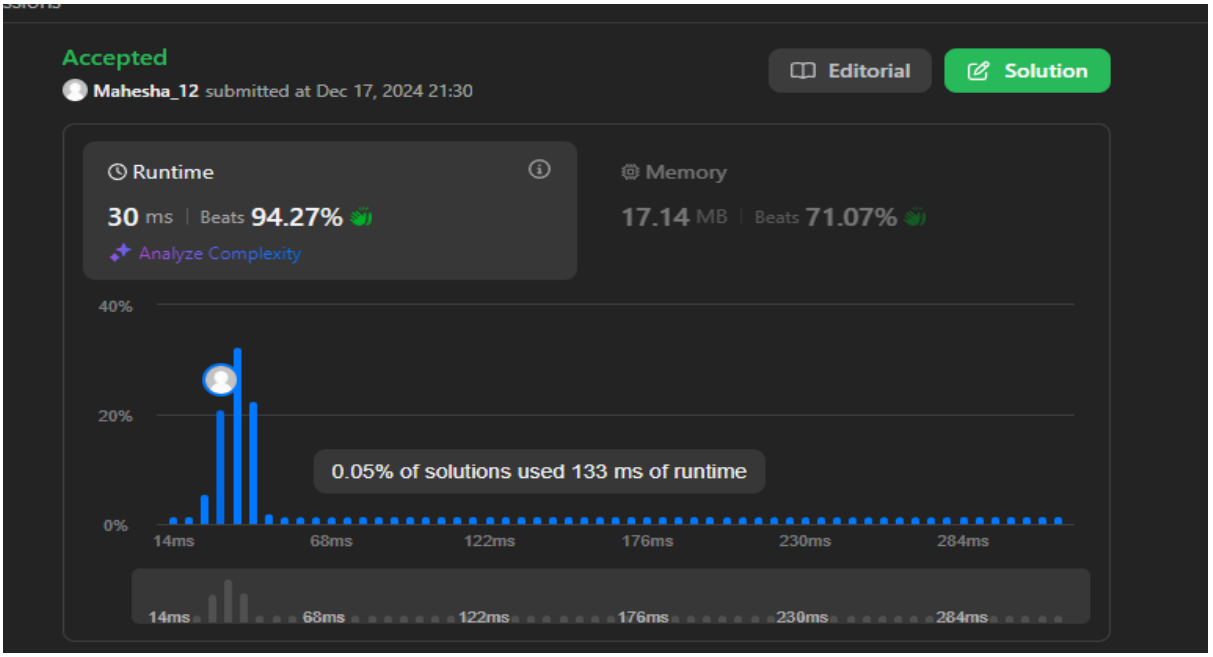
skipB =
3

Output

```
Intersected at '8'
```

Expected

```
Intersected at '8'
```



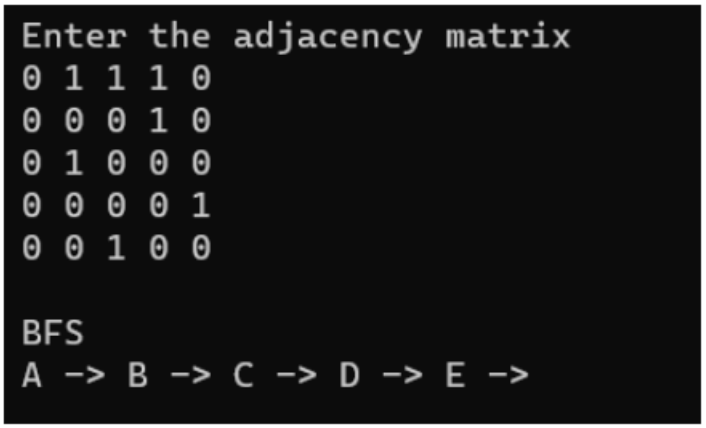
LAB PROGRAM 9

A] Write a program to traverse a graph using BFS method.

```
#include <stdio.h>
#define MAX 5
void bfs(int adj[][MAX], int visited[], int start) {
 int q[MAX], front = -1, rear = -1, i;
 for (i = 0; i < MAX; i++)
 visited[i] = 0;
 q[++rear] = start;
 ++front;
 visited[start] = 1;
 while (rear >= front) {
 start = q[front++];
 printf("%c -> ", start + 'A');
 for (i = 0; i < MAX; i++) {
 if (adj[start][i] && visited[i] == 0) {
 q[++rear] = i;
 visited[i] = 1;
 }
 }
 }
 printf("\n");
}
int main() {
 int adj[MAX][MAX], visited[MAX], i, j;
 printf("Enter the adjacency matrix\n");
 for (i = 0; i < MAX; i++) {
```

```
 for (j = 0; j < MAX; j++) {
scanf("%d", &adj[i][j]);
 }
 }
 printf("\nBFS\n");
 bfs(adj, visited, 0);
 return 0;
}
```

Output:



B] Write a program to check whether given graph is connected or not using DFS method.

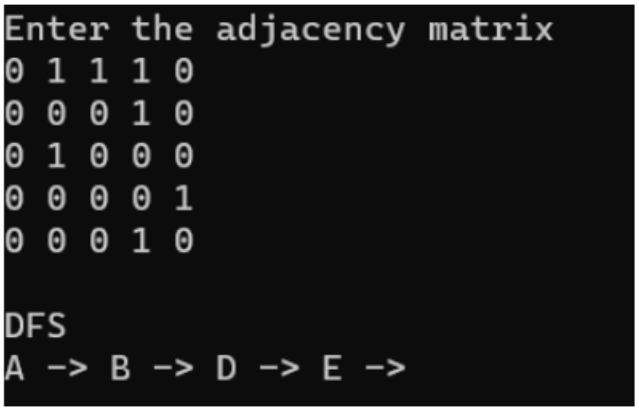
```
#include <stdio.h>
#define MAX 5
void dfs(int adj[][MAX], int visited[], int start) {
 int s[MAX], top = -1, i;
 for (i = 0; i < MAX; i++)
 visited[i] = 0;
 s[++top] = start;
 visited[start] = 1;
 while (top != -1) {
 start = s[top--];
 printf("%c -> ", start + 'A');
 for (i = 0; i < MAX; i++) {
 if (adj[start][i] && visited[i] == 0) {
 s[++top] = i;
 visited[i] = 1;
 break;
 }
 }
 }
 printf("\n");
}
int main() {
 int adj[MAX][MAX], visited[MAX], i, j;
 printf("Enter the adjacency matrix\n");
 for (i = 0; i < MAX; i++) {
```

```
 for (j = 0; j < MAX; j++) {
```



```
 scanf("%d", &adj[i][j]);
 }
 }
 printf("\nDFS\n");
 dfs(adj, visited, 0);
 return 0;
}
```

Output:



LAB PROGRAM 10

Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F. 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. Let the keys in K and addresses in L are integers. Design and develop a Program in C that uses Hash function H: K -> L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

```
#include <stdio.h>

#include <stdlib.h>

int key[20], n, m;

int *ht;

int count = 0;

void insert(int key) {

 int index = key % m;

 while (ht[index] != -1) {

 index = (index + 1) % m;

 }

 ht[index] = key;

 count++;

}

void display() {

 int i;

 if (count == 0) {

 printf("\nHash Table is empty");

 return;

 }

 printf("\nHash Table contents are:\n");

 for (i = 0; i < m; i++) {
```

```
 printf("\n T[%d] --> %d", i, ht[i]);
 }
}
int main() {
 int i;
 printf("\nEnter the number of employee records (N): ");
 scanf("%d", &n);
 printf("\nEnter the memory size (m) for the hash table: ");
 scanf("%d", &m);
 ht = (int *)malloc(m * sizeof(int));
 if (!ht) {
 printf("\nMemory allocation failed.");
 return 1;
 }
 for (i = 0; i < m; i++) {
 ht[i] = -1;
 }
 printf("\nEnter the four-digit key values (K) for %d Employee Records:\n", n);
 for (i = 0; i < n; i++) {
 scanf("%d", &key[i]);
 }
 for (i = 0; i < n; i++) {
 if (count == m) {
 printf("\nHash table is full. Cannot insert the record for key %d", key[i]);
 break;
 }
 insert(key[i]);
```

```
}
display();
return 0;
}
```

Output:

```
Enter the number of employee records (N): 5

Enter the memory size (m) for the hash table: 7

Enter the four-digit key values (K) for 5 Employee Records:
4256
7895
1245
2000
8954

Hash Table contents are:

 T[0] --> 4256
 T[1] --> 1245
 T[2] --> 8954
 T[3] --> -1
 T[4] --> -1
 T[5] --> 2000
 T[6] --> 7895
Process returned 0 (0x0)   execution time : 33.698 s
Press any key to continue.
```