—

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

G M KUSUMA (1BM24CS405)

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 G M KUSUMA 24BECS402, who is 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.

Dr. Selva kumar S
Associate 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>
#define STACK_SIZE 5
void push(int st[],int *top)
{
	int item;
	if(*top==STACK_SIZE-1)
		printf("Stack overflow\n");
	else
	{
		printf("\nEnter an item :");
		scanf("%d",&item);
		(*top)++;
		st[*top]=item;
	}
}
void pop(int st[],int *top)
{
	if(*top==-1)
		printf("Stack underflow\n");
	else
	{
		printf("\n%d item was deleted",st[(*top)--]);
	}
}
void display(int st[],int *top)
{
	int i;
	if(*top==-1)
		printf("Stack is empty\n");
	for(i=0;i<=*top;i++)
		printf("%d\t",st[i]);
}
void main()
{
	int st[10],top=-1, c,val_del;
	while(1)
	{
		printf("\n1. Push\n2. Pop\n3. Display\n");
		printf("\nEnter your choice :");
		scanf("%d",&c);
		switch(c)
		{
```

```
                    case 1: push(st,&top);
                            break;
                    case 2: pop(st,&top);
                            break;
                    case 3: display(st,&top);
                            break;
                    default: printf("\nInvalid choice!!!");
                            exit(0);
                }
        }
}
```

Output:

```
PS D:\kusumaDST> .\LAB1
1.push
 2.pop
 3.display
enter choice: 1
enter element: 10
1.push
 2.pop
 3.display
enter choice: 1
enter element: 20
1.push
 2.pop
 3.display
enter choice: 1
slack overflow
1.push
 2.pop
 3.display
enter choice: 3
content in the slack:
1020
1.push
 2.pop
 3.display
enter choice: 2
popped elements are: 20
1.push
 2.pop
 3.display
enter choice: 4
PS D:\kusumaDST>
```

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 <string.h>

int infixIndex = 0, pos = 0, top = -1, length;
char symbol, temp, infix[20], postfix[20], stack[20];
void infixtopostfix();
void push(char symbol);
char pop();
int pred(char symbol);
void main() {
   printf("Enter infix expression:\n");
   scanf("%s", infix);

   length = strlen(infix); // Set length of the infix expression
   push('#');  // Push a sentinel character onto the stack
while (infixIndex < length) {
      symbol = infix[infixIndex];

      switch (symbol) {
         case '(':
            push(symbol);
            break;
       case ')':
            temp = pop();
            while (temp != '(') {
               postfix[pos++] = temp;
               temp = pop();
            }
            break;

         case '+':  // Operator case for +, -, *, /
         case '-':
         case '*':
         case '/':
            while (pred(stack[top]) >= pred(symbol)) {
               temp = pop();
               postfix[pos++] = temp;}
            push(symbol);
            break;

         default:  // Operand case (e.g., a variable or number)
            postfix[pos++] = symbol;
            break;
      }
      infixIndex++;
```

```
    }

    // Pop remaining operators from the stack
    while (top > 0) {
        temp = pop();
        postfix[pos++] = temp; }

    postfix[pos] = '\0';  // Null terminate the postfix expression

    printf("\nInfix expression: %s", infix);
    printf("\nPostfix expression: %s", postfix);}

void push(char symbol) {
    top = top + 1;
    stack[top] = symbol;
}

char pop() {
    char symb;
    symb = stack[top];
    top = top - 1;
    return symb;
}int pred(char symbol) {
    int p;
    switch (symbol) {
        case '*':
        case '/': p = 2; break;
        case '+':
        case '-': p = 1; break;
        case '(': p = 0; break;
        case '#': p = -1; break;
        default: p = 0; break;  // In case of unexpected symbol
    }
    return p;
}
```

output:

```
PS D:\kusumaDST> cd "d:\kusumaDST\" ; if ($?) { gcc LAB2.c -o LAB2 } ; if
Enter infix expression:
(a*b)*c+d

Infix expression: (a*b)*c+d
Postfix expression: ab*c*d+
PS D:\kusumaDST>
```

3.a) 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

code:

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

#define MAX 50

void insert();
void delete();
void display();

int queue_array[MAX];
int rear = -1;
int front = -1;

int main()
{
   int choice;
   while (1)
   {
      printf("1. Insert element to queue \n");
      printf("2. Delete element from queue \n");
      printf("3. Display all elements of queue \n");
      printf("4. Quit \n");
      printf("Enter your choice: ");
      scanf("%d", &choice);
      switch (choice)
      {
         case 1:
            insert();
            break;
         case 2:
            delete();
            break;
         case 3:
            display();
            break;
         case 4:
            exit(0);
         default:
            printf("Wrong choice \n");
      }
   }
}

void insert()
{
   int add_item;
```

```
    if (rear == MAX - 1)
        printf("Queue Overflow \n");
    else
    {
        if (front == -1)
            /* If queue is initially empty */
            front = 0;
        printf("Insert the element in queue: ");
        scanf("%d", &add_item);
        rear = rear + 1;
        queue_array[rear] = add_item;
    }
}

void delete()
{
    if (front == -1 || front > rear)
    {
        printf("Queue Underflow \n");
        return;
    }
    else
    {
        printf("Element deleted from queue is: %d\n", queue_array[front]);
        front = front + 1;
    }
}

void 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");
    }
}
```

output:

```
PS D:\kusumaDST> cd "d:\kusumaDST\" ; if ($?) { gcc LAB3.c -o LAB3 } ; if ($?) { .\LAB3 }
1. Insert element to queue
2. Delete element from queue
3. Display all elements of queue
4. Quit
Enter your choice: 1
Insert the element in queue: 10
1. Insert element to queue
2. Delete element from queue
3. Display all elements of queue
4. Quit
Enter your choice: 1
Insert the element in queue: 20
1. Insert element to queue
2. Delete element from queue
3. Display all elements of queue
4. Quit
Enter your choice: 1
Insert the element in queue: 25
1. Insert element to queue
2. Delete element from queue
3. Display all elements of queue
4. Quit
Enter your choice: 1
Insert the element in queue: 30
1. Insert element to queue
2. Delete element from queue
3. Display all elements of queue
4. Quit
Enter your choice: 1
Insert the element in queue: 40
1. Insert element to queue
2. Delete element from queue
3. Display all elements of queue
4. Quit
Enter your choice: 3
Queue is:
10 20 25 30 40
1. Insert element to queue
2. Delete element from queue
3. Display all elements of queue
4. Quit
Enter your choice: 2
Element deleted from queue is: 10
1. Insert element to queue
2. Delete element from queue
3. Display all elements of queue
4. Quit
Enter your choice: 3
Queue is:
20 25 30 40
1. Insert element to queue
2. Delete element from queue
3. Display all elements of queue
4. Quit
Enter your choice: 4
PS D:\kusumaDST>
```

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

code:

```
#include <stdio.h>
#define MAX 5

int queue[MAX];
int front = -1, rear = -1;

void insert() {
   int element;
   if ((rear + 1) % MAX == front) {
      printf("Queue Overflow! Cannot insert element.\n");
      return;
   }
   printf("Enter the element to insert: ");
   scanf("%d", &element);

   if (front == -1) {
      front = rear = 0;
   } else {
      rear = (rear + 1) % MAX;
   }

   queue[rear] = element;  // Corrected placement of this line
   printf("Inserted %d into the queue.\n", element);
}

void delete() {
   if (front == -1) {
      printf("Queue Underflow! Queue is empty.\n");
      return;
   }

   printf("Deleted element: %d\n", queue[front]);
   if (front == rear) {
      front = rear = -1;
   } else {
      front = (front + 1) % MAX;
   }
}

void display() {
   if (front == -1) {
      printf("Queue is empty.\n");
      return;
   }
```

```
    printf("Queue elements: ");
    int i = front;
    while (i != rear) {
        printf("%d ", queue[i]);
        i = (i + 1) % MAX;
    }
    printf("%d\n", queue[rear]);
}int main() {
    int choice; do {
        printf("\nCircular Queue Operations:\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:
              insert();
              break;

  case 2:
              delete();
              break;

 case 3:
              display();
              break;

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

    return 0;
}
```

output:

```
PS D:\kusumaDST> cd "d:\kusumaDST\" ; if ($?) { gcc LAB4.c -o LAB4 } ; if ($?) {

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

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the element to insert: 20
Inserted 20 into the queue.

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the element to insert: 30
Inserted 30 into the queue.

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the element to insert: 40
Inserted 40 into the queue.
```

```
Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements: 10 20 30 40 50

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted element: 10

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements: 20 30 40 50

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 4
Exiting...
PS D:\kusumaDST>
```

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

code:

```
#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;
}

// Function to insert a node at the end
void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->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* temp = *head;
    for (int i = 0; temp != NULL && i < position - 1; i++) {
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Position is greater than the length of the list.\n");
        free(newNode);
    } else {
        newNode->next = temp->next;
        temp->next = newNode;
    }
}

// Function to display the linked list
void displayList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}


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

    do {

        printf("\n1. Insert at First\n");
        printf("2. Insert at End\n");
        printf("3. Insert at Position\n");
        printf("4. Display List\n");
        printf("5. 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 position and data to insert: ");
            scanf("%d %d", &position, &data);
            insertAtPosition(&head, data, position);
            break;
        case 4:
            printf("Linked List: ");
            displayList(head);
            break;
        case 5:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice! Please try again.\n");
    }
  } while (choice != 5);

  return 0;
}
```

output:

```
PS D:\kusumaDST> cd "d:\kusumaDST\" ; if ($?) { gcc LAB4.c -o LAB4 } ; if ($?)

1. Insert at First
2. Insert at End
3. Insert at Position
4. Display List
5. Exit
Enter your choice: 1
Enter data to insert at first: 10

1. Insert at First
2. Insert at End
3. Insert at Position
4. Display List
5. Exit
Enter your choice: 1
Enter data to insert at first: 20

1. Insert at First
2. Insert at End
3. Insert at Position
4. Display List
5. Exit
Enter your choice: 1
Enter data to insert at first: 30

1. Insert at First
2. Insert at End
3. Insert at Position
4. Display List
5. Exit
Enter your choice: 1
Enter data to insert at first: 35
```

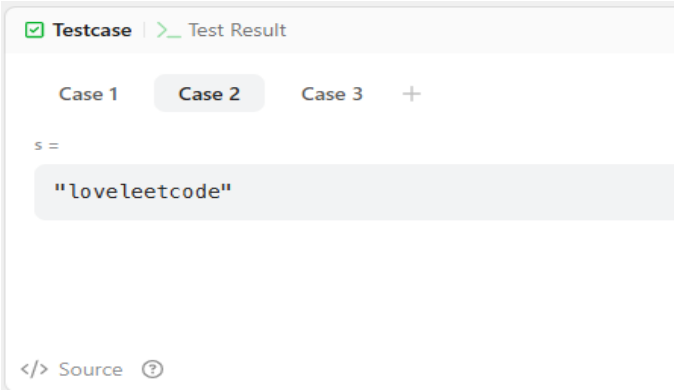
```
1. Insert at First
2. Insert at End
3. Insert at Position
4. Display List
5. Exit
Enter your choice: 1
Enter data to insert at first: 35

1. Insert at First
2. Insert at End
3. Insert at Position
4. Display List
5. Exit
Enter your choice: 2
Enter data to insert at end: 40

1. Insert at First
2. Insert at End
3. Insert at Position
4. Display List
5. Exit
Enter your choice: 4
Linked List: 35 -> 30 -> 20 -> 10 -> 40 -> NULL

1. Insert at First
2. Insert at End
3. Insert at Position
4. Display List
5. Exit
Enter your choice: 2
Enter data to insert at end: 45

1. Insert at First
2. Insert at End
3. Insert at Position
4. Display List
5. Exit
Enter your choice: 4
```

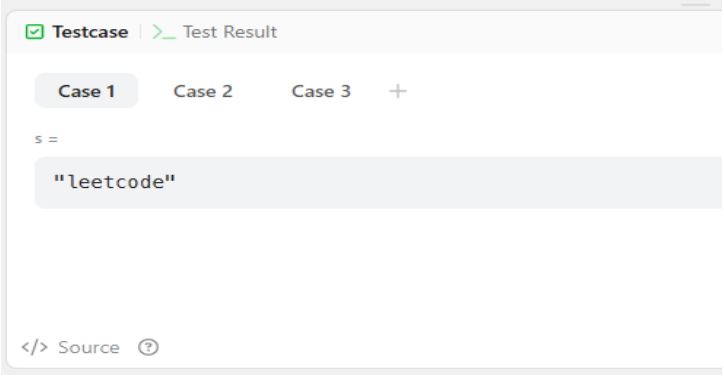
```
5. Exit
Enter your choice: 4
Linked List: 35 -> 30 -> 20 -> 10 -> 40 -> 45 -> NULL

1. Insert at First
2. Insert at End
3. Insert at Position
4. Display List
5. Exit
Enter your choice: 3
Enter position and data to insert: 20

25
Position is greater than the length of the list.

1. Insert at First
2. Insert at End
3. Insert at Position
4. Display List
5. Exit
Enter your choice: 4
Linked List: 35 -> 30 -> 20 -> 10 -> 40 -> 45 -> NULL

1. Insert at First
2. Insert at End
3. Insert at Position
4. Display List
5. Exit
Enter your choice: 3
Enter position and data to insert: 20
15
Position is greater than the length of the list.

1. Insert at First
2. Insert at End
3. Insert at Position
4. Display List
5. Exit
Enter your choice:
```

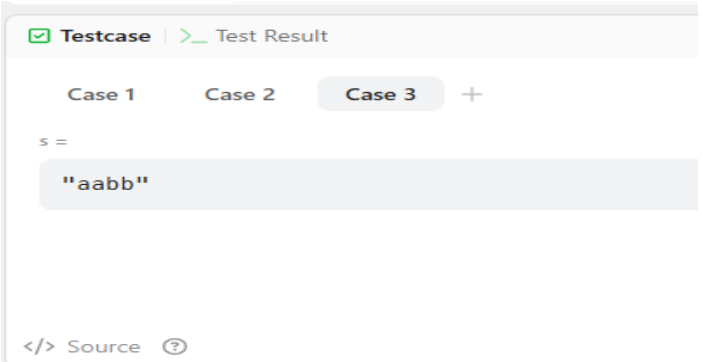
Leetcode
code:

```c
int firstUniqChar(char* s) {
    int k = strlen(s);
    for (int i = 0; i < k; i++) {
        int flag = 1;
        if (s[i] != '#') {
            for (int j = i + 1; j < k; j++) {
                if (s[i] == s[j]) {
                    s[j] = '#';
                    flag = 0;
                }
            }
            if (flag == 1) {
                return i;
            }
        }
    }
    return -1;
}
```

output:







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.

code:

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

// Define the structure for a node
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}

// Function to display the list
void displayList(struct Node* head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

// Function to delete from the beginning
void deleteFromBeginning(struct Node** head) {
    if (*head == NULL) {
        printf("List is already empty.\n");
        return;
    }
    struct Node* temp = *head;
    *head = (*head)->next;
    free(temp);
    printf("Node deleted from the beginning.\n");
}

// Function to delete from the end
```

```
void deleteFromEnd(struct Node** head) {
    if (*head == NULL) {
        printf("List is already empty.\n");
        return;
    }
    if ((*head)->next == NULL) {
        free(*head);
        *head = NULL;
        printf("Node deleted from the end.\n");
        return;
    }
    struct Node* temp = *head;
    while (temp->next && temp->next->next) {
        temp = temp->next;
    }
    free(temp->next);
    temp->next = NULL;
    printf("Node deleted from the end.\n");
}

// Function to delete a node from a specific position
void deleteAtPosition(struct Node** head, int position) {
    if (*head == NULL) {
        printf("List is empty.\n");
        return;
    }
    if (position == 0) {
        deleteFromBeginning(head);
        return;
    }

    struct Node* temp = *head;
    for (int i = 0; temp != NULL && i < position - 1; i++) {
        temp = temp->next;
    }

    if (temp == NULL || temp->next == NULL) {
        printf("Position out of range.\n");
        return;
    }

    struct Node* nodeToDelete = temp->next;
    temp->next = temp->next->next;
    free(nodeToDelete);
    printf("Node deleted from position %d.\n", position);
}

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

```
while (1) {
    printf("\n1. Create Node\n");
    printf("2. Display List\n");
    printf("3. Delete Node from Beginning\n");
    printf("4. Delete Node from End\n");
    printf("5. Delete Node from Specific Position\n");
    printf("6. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter value to insert: ");
            scanf("%d", &value);
            if (head == NULL) {
                head = createNode(value);
            } else {
                struct Node* temp = head;
                while (temp->next != NULL) {
                    temp = temp->next;
                }
                temp->next = createNode(value);
            }
            break;

        case 2:
            displayList(head);
            break;

        case 3:
            deleteFromBeginning(&head);
            break;

        case 4:
            deleteFromEnd(&head);
            break;

        case 5:
            printf("Enter position to delete: ");
            scanf("%d", &position);
            deleteAtPosition(&head, position);
            break;

        case 6:
            printf("Exiting program.\n");
            return 0;

        default:
            printf("Invalid choice, please try again.\n");} }
```

```
    return 0;
}
}
```

output:

```
PS D:\kusumaDST> cd "d:\kusumaDST\" ; if ($?) { gcc LAB5.c -o LAB5 } ; if ($?)

1. Create Node
2. Display List
3. Delete Node from Beginning
4. Delete Node from End
5. Delete Node from Specific Position
6. Exit
Enter your choice: 1
Enter value to insert: 10

1. Create Node
2. Display List
3. Delete Node from Beginning
4. Delete Node from End
5. Delete Node from Specific Position
6. Exit
Enter your choice: 1
Enter value to insert: 20

1. Create Node
2. Display List
3. Delete Node from Beginning
4. Delete Node from End
5. Delete Node from Specific Position
6. Exit
Enter your choice: 1
Enter value to insert: 25

1. Create Node
2. Display List
3. Delete Node from Beginning
4. Delete Node from End
5. Delete Node from Specific Position
6. Exit
Enter your choice: 1
Enter value to insert: 30
```

:

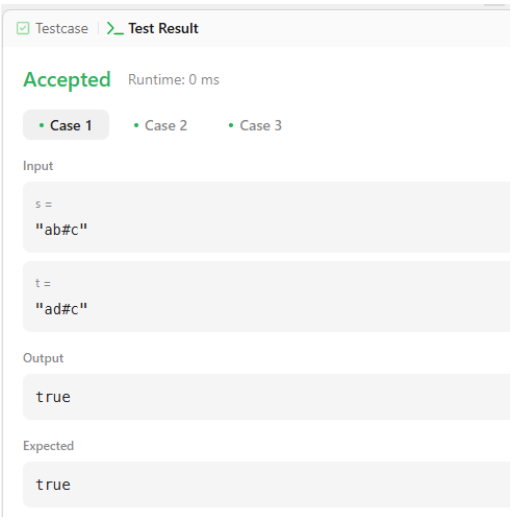
```
1. Create Node
2. Display List
3. Delete Node from Beginning
4. Delete Node from End
5. Delete Node from Specific Position
6. Exit
Enter your choice: 3
Node deleted from the beginning.

1. Create Node
2. Display List
3. Delete Node from Beginning
4. Delete Node from End
5. Delete Node from Specific Position
6. Exit
Enter your choice: 2
20 -> 25 -> 30 -> 40 -> 50 -> NULL

1. Create Node
2. Display List
3. Delete Node from Beginning
4. Delete Node from End
5. Delete Node from Specific Position
6. Exit
Enter your choice: 4
Node deleted from the end.

1. Create Node
2. Display List
3. Delete Node from Beginning
4. Delete Node from End
5. Delete Node from Specific Position
6. Exit
Enter your choice: 2
20 -> 25 -> 30 -> 40 -> NULL
```

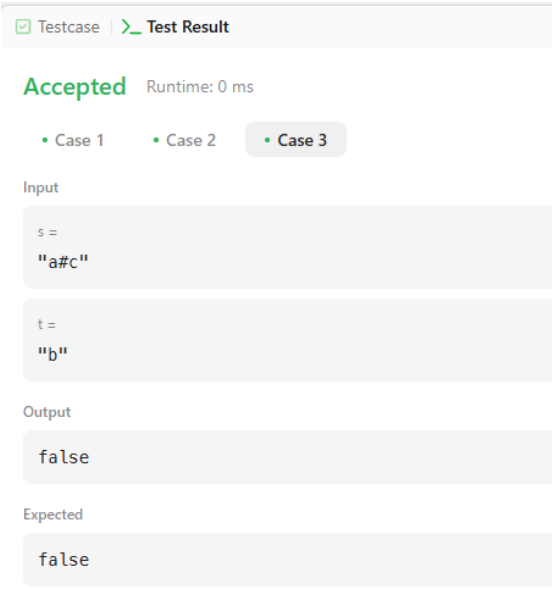
```
1. Create Node
2. Display List
3. Delete Node from Beginning
4. Delete Node from End
5. Delete Node from Specific Position
6. Exit
Enter your choice: 5
Enter position to delete: 2
Node deleted from position 2.

1. Create Node
2. Display List
3. Delete Node from Beginning
4. Delete Node from End
5. Delete Node from Specific Position
6. Exit
Enter your choice: 2
20 -> 25 -> 40 -> NULL

1. Create Node
2. Display List
3. Delete Node from Beginning
4. Delete Node from End
5. Delete Node from Specific Position
6. Exit
Enter your choice: 6
Exiting program.
PS D:\kusumaDST>
```

Leetcode
code:

```
bool backspaceCompare(char* s, char* t) {
int i = strlen(s) - 1, j = strlen(t) - 1;
while (i >= 0 || j >= 0) {
int skip_s = 0;
while (i >= 0 && (s[i] == '#' || skip_s > 0)) {
if (s[i] == '#') {skip_s++;} else {
skip_s--;}i--;}
int skip_t = 0;
while (j >= 0 && (t[j] == '#' || skip_t > 0)) {
if (t[j] == '#') {skip_t++;} else {skip_t--;}j--;}
if (i >= 0 && j >= 0 && s[i] != t[j]) {return false;}
if ((i >= 0) != (j >= 0)) {
return false;}i--;j--;return true;}
```

output:



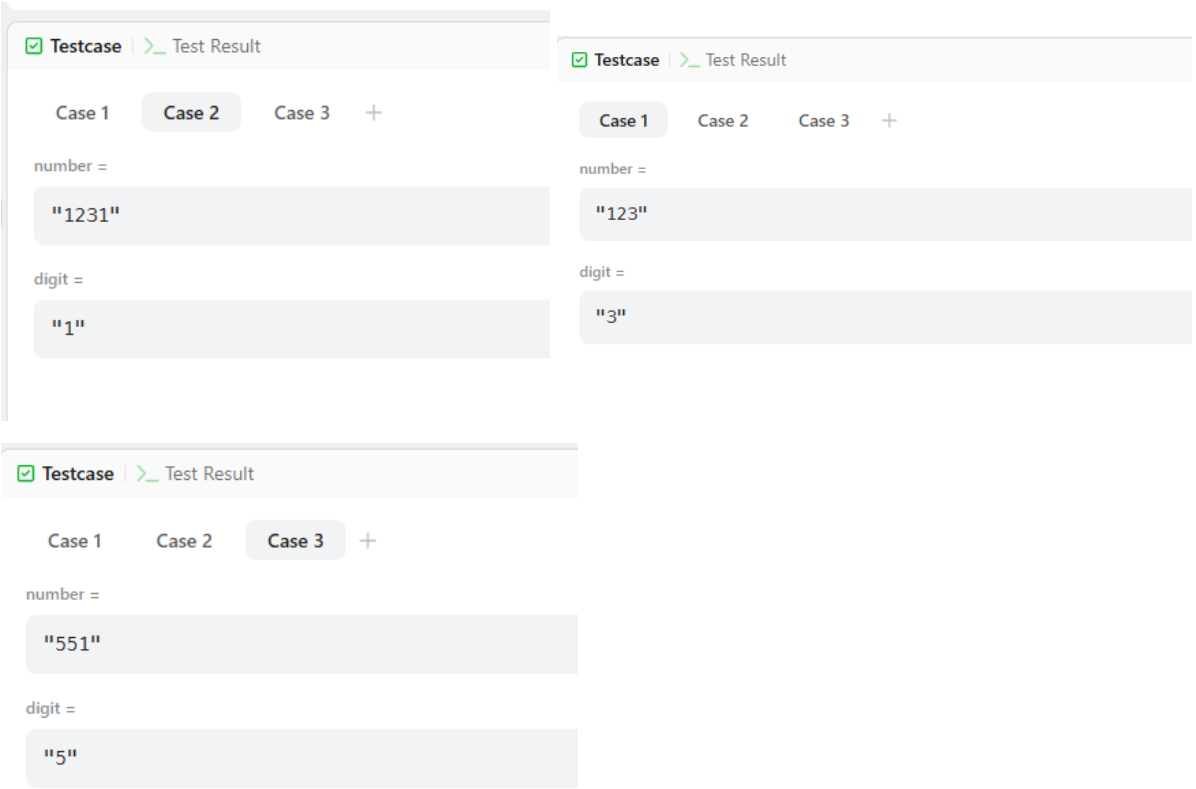


Leetcode

code

```
#include<string.h>
char* removeDigit(char* number, char digit) {
    int i,j=-1,l=strlen(number);
for(i=0;i<l;i++){
        if(number[i]==digit){
           j=i;if(i<l-1&&number[i]<number[i+1]){break}
   for (int i = j; i < l - 1; i++) {
        number[i] = number[i + 1]}
    number[l - 1] = '\0';
    return number;}
```

output:



6a) WAP to Implement Single Link List with following operations: Sortthelinkedlist, Reversethelinkedlist, Concatenation of two linked lists.

code:

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

// Define the Node structure
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a node at the end of the list
void insertEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

// Function to print the list
void printList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

// Function to sort the linked list (simple bubble sort)
void sortList(struct Node* head) {
    struct Node *i, *j;
    int temp;
    for (i = head; i != NULL; i = i->next) {
```

```
        for (j = i->next; j != NULL; j = j->next) {
            if (i->data > j->data) {
                // Swap data
                temp = i->data;
                i->data = j->data;
                j->data = temp;
            }
        }
    }
}

// Function to reverse the linked list
void reverseList(struct Node** head) {
    struct Node *prev = NULL, *current = *head, *next = NULL;
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head = prev;
}

// Function to concatenate two lists
void concatenateLists(struct Node** head1, struct Node* head2) {
    if (*head1 == NULL) {
        *head1 = head2;
    } else {
        struct Node* temp = *head1;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = head2;
    }
}

// Main function
int main() {
    struct Node* list1 = NULL;
    struct Node* list2 = NULL;
    int choice, data;

    while(1) {
        printf("\n1. Insert into List 1\n");
        printf("2. Insert into List 2\n");
        printf("3. Print List 1\n");
        printf("4. Print List 2\n");
        printf("5. Sort List 1\n");
        printf("6. Reverse List 1\n");
        printf("7. Concatenate List 1 and List 2\n");
```

```
        printf("8. Exit\n");

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

        switch(choice) {
            case 1:
                printf("Enter data to insert into List 1: ");
                scanf("%d", &data);
                insertEnd(&list1, data);
                break;
            case 2:
                printf("Enter data to insert into List 2: ");
                scanf("%d", &data);
                insertEnd(&list2, data);
                break;
            case 3:
                printf("List 1: ");
                printList(list1);
                break;
            case 4:
                printf("List 2: ");
                printList(list2);
                break;
            case 5:
                sortList(list1);
                printf("List 1 after sorting: ");
                printList(list1);
                break;
            case 6:
                reverseList(&list1);
                printf("List 1 after reversing: ");
                printList(list1);
                break;
            case 7:
                concatenateLists(&list1, list2);
                printf("List 1 after concatenation: ");
                printList(list1);
                break;
            case 8:
                exit(0);
                break;
            default:
                printf("Invalid choice! Please try again.\n");}} }
 return 0;
}
```

output:

```
PS D:\kusumaDST> cd "d:\kusumaDST\" ; if ($?) { gcc LAB6.c -o LAB6 } ; if ($?)

1. Insert into List 1
2. Insert into List 2
3. Print List 1
4. Print List 2
5. Sort List 1
6. Reverse List 1
7. Concatenate List 1 and List 2
8. Exit
Enter your choice: 1
Enter data to insert into List 1: 10

1. Insert into List 1
2. Insert into List 2
3. Print List 1
4. Print List 2
5. Sort List 1
6. Reverse List 1
7. Concatenate List 1 and List 2
8. Exit
Enter your choice: 1
Enter data to insert into List 1: 20

1. Insert into List 1
2. Insert into List 2
3. Print List 1
4. Print List 2
5. Sort List 1
6. Reverse List 1
7. Concatenate List 1 and List 2
8. Exit
Enter your choice: 1
Enter data to insert into List 1: 30
```

```
1. Insert into List 1
2. Insert into List 2
3. Print List 1
4. Print List 2
5. Sort List 1
6. Reverse List 1
7. Concatenate List 1 and List 2
8. Exit
Enter your choice: 1
Enter data to insert into List 1: 40

1. Insert into List 1
2. Insert into List 2
3. Print List 1
4. Print List 2
5. Sort List 1
6. Reverse List 1
7. Concatenate List 1 and List 2
8. Exit
Enter your choice: 1
Enter data to insert into List 1: 50

1. Insert into List 1
2. Insert into List 2
3. Print List 1
4. Print List 2
5. Sort List 1
6. Reverse List 1
7. Concatenate List 1 and List 2
8. Exit
Enter your choice: 3
List 1: 10 -> 20 -> 30 -> 40 -> 50 -> NULL

1. Insert into List 1
2. Insert into List 2
3. Print List 1
4. Print List 2
5. Sort List 1
6. Reverse List 1
```

```
4. Print List 2
5. Sort List 1
6. Reverse List 1
7. Concatenate List 1 and List 2
8. Exit
Enter your choice: 6
List 1 after reversing: 50 -> 40 -> 30 -> 20 -> 10 -> NULL

1. Insert into List 1
2. Insert into List 2
3. Print List 1
4. Print List 2
5. Sort List 1
6. Reverse List 1
7. Concatenate List 1 and List 2
8. Exit
Enter your choice: 7
List 1 after concatenation: 50 -> 40 -> 30 -> 20 -> 10 -> 5 -> 15 -> 25 -> 35 -> 45 -> 55 -> NULL

1. Insert into List 1
2. Insert into List 2
3. Print List 1
4. Print List 2
5. Sort List 1
6. Reverse List 1
7. Concatenate List 1 and List 2
8. Exit
Enter your choice: 6
List 1 after reversing: 55 -> 45 -> 35 -> 25 -> 15 -> 5 -> 10 -> 20 -> 30 -> 40 -> 50 -> NULL

1. Insert into List 1
2. Insert into List 2
3. Print List 1
4. Print List 2
5. Sort List 1
6. Reverse List 1
7. Concatenate List 1 and List 2
8. Exit
Enter your choice:
```

6 b) WAP to Implement Single Link List to simulate Stack & Queue Operations.

code:

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

// Definition of a Node
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// ================== Stack Operations ====================
```

```
// Push operation for Stack
void push(struct Node** top, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *top;
    *top = newNode;
    printf("Pushed %d to stack\n", data);
}

// Pop operation for Stack
int pop(struct Node** top) {
    if (*top == NULL) {
        printf("Stack is empty!\n");
        return -1; // Return -1 if stack is empty
    }
    struct Node* temp = *top;
    int data = temp->data;
    *top = (*top)->next;
    free(temp);
    printf("Popped %d from stack\n", data);
    return data;
}

// Display operation for Stack
void displayStack(struct Node* top) {
    if (top == NULL) {
        printf("Stack is empty!\n");
        return;
    }
    printf("Stack: ");
    struct Node* temp = top;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

// Check if the Stack is empty
int isEmptyStack(struct Node* top) {
    return top == NULL;
}

// ================= Queue Operations ====================

// Enqueue operation for Queue
void enqueue(struct Node** front, struct Node** rear, int data) {
    struct Node* newNode = createNode(data);
    if (*rear == NULL) {
        *front = *rear = newNode;
        printf("Enqueued %d to queue\n", data);
```

```
        return;
    }
    (*rear)->next = newNode;
    *rear = newNode;
    printf("Enqueued %d to queue\n", data);
}

// Dequeue operation for Queue
int dequeue(struct Node** front) {
    if (*front == NULL) {
        printf("Queue is empty!\n");
        return -1; // Return -1 if queue is empty
    }
    struct Node* temp = *front;
    int data = temp->data;
    *front = (*front)->next;
    free(temp);
    printf("Dequeued %d from queue\n", data);
    return data;
}

// Display operation for Queue
void displayQueue(struct Node* front) {
    if (front == NULL) {
        printf("Queue is empty!\n");
        return;
    }
    printf("Queue: ");
    struct Node* temp = front;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

// Front operation for Queue
int frontQueue(struct Node* front) {
    if (front == NULL) {
        printf("Queue is empty!\n");
        return -1;
    }
    return front->data;
}

// Check if the Queue is empty
int isEmptyQueue(struct Node* front) {
    return front == NULL;
}
```

```
// ================ Main Function ========================

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

    while (1) {
        printf("\n1. Stack Operations\n");
        printf("2. Queue Operations\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: // Stack operations
                while (1) {
                    printf("\nStack Operations:\n");
                    printf("1. Push\n");
                    printf("2. Pop\n");
                    printf("3. Display\n");
                    printf("4. Check if Stack is Empty\n");
                    printf("5. Back to Main Menu\n");
                    printf("Enter your choice: ");
                    scanf("%d", &choice);

                    switch (choice) {
                        case 1:
                            printf("Enter data to push: ");
                            scanf("%d", &data);
                            push(&stackTop, data);
                            break;
                        case 2:
                            pop(&stackTop);
                            break;
                        case 3:
                            displayStack(stackTop);
                            break;
                        case 4:
                            if (isEmptyStack(stackTop)) {
                                printf("Stack is empty\n");
                            } else {
                                printf("Stack is not empty\n");
                            }
                            break;
                        case 5:
                            goto stackMenu;
                        default:
                            printf("Invalid choice! Please try again.\n");
```

```
        }
    }
stackMenu:

case 2: // Queue operations
    while (1) {
        printf("\nQueue Operations:\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Display\n");
        printf("4. Front\n");
        printf("5. Check if Queue is Empty\n");
        printf("6. Back to Main Menu\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to enqueue: ");
                scanf("%d", &data);
                enqueue(&queueFront, &queueRear, data);
                break;
            case 2:
                dequeue(&queueFront);
                break;
            case 3:
                displayQueue(queueFront);
                break;
            case 4:
                printf("Front element is: %d\n", frontQueue(queueFront));
                break;
            case 5:
                if (isEmptyQueue(queueFront)) {
                    printf("Queue is empty\n");
                } else {
                    printf("Queue is not empty\n");
                }
                break;
            case 6:
                goto queueMenu;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }
queueMenu:

case 3:
    exit(0);
    break;
default:
```

```
            printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}
```

output:

```
PS D:\kusumaDST> cd "d:\kusumaDST\" ; if ($?) { gcc LAB6b.c -o LAB6b } ; if ($?)

1. Stack Operations
2. Queue Operations
3. Exit
Enter your choice: 1

Stack Operations:
1. Push
2. Pop
3. Display
4. Check if Stack is Empty
5. Back to Main Menu
Enter your choice: 1
Enter data to push: 10
Pushed 10 to stack

Stack Operations:
1. Push
2. Pop
3. Display
4. Check if Stack is Empty
5. Back to Main Menu
Enter your choice: 1
Enter data to push: 20
Pushed 20 to stack

Stack Operations:
1. Push
2. Pop
3. Display
4. Check if Stack is Empty
5. Back to Main Menu
Enter your choice: 1
Enter data to push: 30
Pushed 30 to stack
```

```
Stack Operations:
1. Push
2. Pop
3. Display
4. Check if Stack is Empty
5. Back to Main Menu
Enter your choice: 1
Enter data to push: 40
Pushed 40 to stack

Stack Operations:
1. Push
2. Pop
3. Display
4. Check if Stack is Empty
5. Back to Main Menu
Enter your choice: 1
Enter data to push: 50
Pushed 50 to stack

Stack Operations:
1. Push
2. Pop
3. Display
4. Check if Stack is Empty
5. Back to Main Menu
Enter your choice: 3
Stack: 50 40 30 20 10

Stack Operations:
1. Push
2. Pop
3. Display
4. Check if Stack is Empty
5. Back to Main Menu
Enter your choice: 2
Popped 50 from stack
```

```
Stack Operations:
1. Push
2. Pop
3. Display
4. Check if Stack is Empty
5. Back to Main Menu
Enter your choice: 3
Stack: 40 30 20 10

Stack Operations:
1. Push
2. Pop
3. Display
4. Check if Stack is Empty
5. Back to Main Menu
Enter your choice: 4
Stack is not empty

Stack Operations:
1. Push
2. Pop
3. Display
4. Check if Stack is Empty
5. Back to Main Menu
Enter your choice: 5
```

```
Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Front
5. Check if Queue is Empty
6. Back to Main Menu
Enter your choice: 1
Enter data to enqueue: 5
Enqueued 5 to queue

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Front
5. Check if Queue is Empty
6. Back to Main Menu
Enter your choice: 1
Enter data to enqueue: 10
Enqueued 10 to queue

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Front
5. Check if Queue is Empty
6. Back to Main Menu
Enter your choice: 1
Enter data to enqueue: 15
Enqueued 15 to queue
```

```
Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Front
5. Check if Queue is Empty
6. Back to Main Menu
Enter your choice: 1
Enter data to enqueue: 25
Enqueued 25 to queue

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Front
5. Check if Queue is Empty
6. Back to Main Menu
Enter your choice: 1
Enter data to enqueue: 35
Enqueued 35 to queue

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Front
5. Check if Queue is Empty
6. Back to Main Menu
Enter your choice: 1
Enter data to enqueue: 45
Enqueued 45 to queue
```

```
Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Front
5. Check if Queue is Empty
6. Back to Main Menu
Enter your choice: 3
Queue: 5 10 15 25 35 45

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Front
5. Check if Queue is Empty
6. Back to Main Menu
Enter your choice: 4
Front element is: 5

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Front
5. Check if Queue is Empty
6. Back to Main Menu
Enter your choice: 2
Dequeued 5 from queue

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Front
5. Check if Queue is Empty
6. Back to Main Menu
Enter your choice: 3
Queue: 10 15 25 35 45
```

```
Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Front
5. Check if Queue is Empty
6. Back to Main Menu
Enter your choice: 3
Queue: 10 15 25 35 45

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Front
5. Check if Queue is Empty
6. Back to Main Menu
Enter your choice: 5
Queue is not empty

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Front
5. Check if Queue is Empty
6. Back to Main Menu
Enter your choice: 6
PS D:\kusumaDST>
```

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.

code:

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

// Define the structure for a Node
struct Node {
    int data;          // Holds the value of the node
    struct Node* prev;  // Points to the previous node
    struct Node* next;  // Points to the next node
};

// Declare the head of the list globally
struct Node* head = NULL;

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

// Function to create a doubly linked list
void createList() {
    int value, choice;
    struct Node* temp;

    do {
        // Prompt user to enter a value
        printf("Enter value to insert: ");
        scanf("%d", &value);

        // Create a new node with the entered value
        struct Node* newNode = createNode(value);

        if (head == NULL) {
            head = newNode;  // If the list is empty, the new node becomes the head
        } else {
            temp = head;
            // Traverse to the last node
            while (temp->next != NULL) {
                temp = temp->next;
            }
```

```
            temp->next = newNode;  // Update the last node's next pointer
            newNode->prev = temp;  // Set the new node's previous pointer to the last node
        }

        // Ask user if they want to add another node
        printf("Do you want to add another node? (1 for Yes, 0 for No): ");
        scanf("%d", &choice);
    } while (choice != 0);
}

// Function to insert a new node to the left of a specific node
void insertLeft(int value, int target) {
    struct Node* temp = head;

    // Search for the node with the target value
    while (temp != NULL && temp->data != target) {
        temp = temp->next;  // Traverse the list to find the target node
    }

    if (temp == NULL) {
        printf("Node with value %d not found.\n", target);
        return;
    }

    // Create a new node to insert
    struct Node* newNode = createNode(value);

    // Insert the new node to the left of the target node
    newNode->next = temp;
    newNode->prev = temp->prev;

    // Update the previous node's next pointer, if it exists
    if (temp->prev != NULL) {
        temp->prev->next = newNode;
    } else {
        head = newNode;  // If inserting at the head, update the head pointer
    }

    // Update the target node's previous pointer
    temp->prev = newNode;

    printf("Node with value %d inserted to the left of %d.\n", value, target);
}

// Function to delete a node based on a specific value
void deleteNode(int value) {
    struct Node* temp = head;

    // Search for the node to delete
    while (temp != NULL && temp->data != value) {
```

```
        temp = temp->next;  // Traverse to find the node with the given value
    }

    if (temp == NULL) {
        printf("Node with value %d not found.\n", value);
        return;
    }

    // If the node has a previous node, update its next pointer
    if (temp->prev != NULL) {
        temp->prev->next = temp->next;
    } else {
        head = temp->next;  // If deleting the head, update the head pointer
    }

    // If the node has a next node, update its previous pointer
    if (temp->next != NULL) {
        temp->next->prev = temp->prev;
    }

    free(temp);  // Free the memory of the deleted node
    printf("Node with value %d deleted.\n", value);
}

// Function to display the contents of the doubly linked list
void displayList() {
    if (head == NULL) {
        printf("List is empty.\n");
        return;00
    }

    struct Node* temp = head;
    // Traverse the list and print each node's data
    while (temp != NULL) {
        printf("%d <-> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

// Main function with a menu-driven approach
int main() {
    int choice, value, target;

    do {
        // Display the menu
        printf("\nDoubly Linked List Operations:\n");
        printf("1. Create a doubly linked list\n");
        printf("2. Insert a new node to the left of a specific node\n");
        printf("3. Delete a node based on specific value\n");
```

```
        printf("4. Display the contents of the list\n");
        printf("5. Exit\n");

        // Ask user for their choice
        printf("Enter your choice: ");
        scanf("%d", &choice);


switch (choice) {
            case 1:
                createList();  // Create a new doubly linked list
                break;

case 2:
                // Ask user for the value to insert and the target node
                printf("Enter value to insert: ");
                scanf("%d", &value);
                printf("Enter target value (left of which node to insert): ");
                scanf("%d", &target);
                insertLeft(value, target);  // Insert the node to the left of the target node
                break;

case 3:
                // Ask user for the value of the node to delete
                printf("Enter value to delete: ");
                scanf("%d", &value);
                deleteNode(value);  // Delete the node with the specified value
                break;

case 4:
                displayList();  // Display the contents of the list
                break;

  case 5:
                printf("Exiting the program.\n");
                break;

default:
                printf("Invalid choice. Please try again.\n");

}
    } while (choice != 5);  // Repeat until the user chooses to exit

    return 0;
}
```

output:

```
PS D:\kusumaDST> cd "d:\kusumaDST\" ; if ($?) { gcc LAB7.c -o LAB7 } ; if

Doubly Linked List Operations:
1. Create a doubly linked list
2. Insert a new node to the left of a specific node
3. Delete a node based on specific value
4. Display the contents of the list
5. Exit
Enter your choice: 1
Enter value to insert: 10
Do you want to add another node? (1 for Yes, 0 for No): 1
Enter value to insert: 20
Do you want to add another node? (1 for Yes, 0 for No): 1
Enter value to insert: 30
Do you want to add another node? (1 for Yes, 0 for No): 1
Enter value to insert: 40
Do you want to add another node? (1 for Yes, 0 for No): 1
Enter value to insert: 50
Do you want to add another node? (1 for Yes, 0 for No): 0

Doubly Linked List Operations:
1. Create a doubly linked list
2. Insert a new node to the left of a specific node
3. Delete a node based on specific value
4. Display the contents of the list
5. Exit
Enter your choice: 4
10 <-> 20 <-> 30 <-> 40 <-> 50 <-> NULL
```

```
Doubly Linked List Operations:
1. Create a doubly linked list
2. Insert a new node to the left of a specific node
3. Delete a node based on specific value
4. Display the contents of the list
5. Exit
Enter your choice: 2
Enter value to insert: 3
Enter target value (left of which node to insert): 20
Node with value 3 inserted to the left of 20.

Doubly Linked List Operations:
1. Create a doubly linked list
2. Insert a new node to the left of a specific node
3. Delete a node based on specific value
4. Display the contents of the list
5. Exit
Enter your choice: 3
Enter value to delete: 3
Node with value 3 deleted.

Doubly Linked List Operations:
1. Create a doubly linked list
2. Insert a new node to the left of a specific node
3. Delete a node based on specific value
4. Display the contents of the list
5. Exit
Enter your choice: 4
10 <-> 20 <-> 30 <-> 40 <-> 50 <-> NULL

Doubly Linked List Operations:
1. Create a doubly linked list
2. Insert a new node to the left of a specific node
3. Delete a node based on specific value
4. Display the contents of the list
5. Exit
Enter your choice: 5
Exiting the program.
PS D:\kusumaDST>
```

leetcode

8 Write a program a) To construct binarySearchtree. b) To traverse the tree using all the methods i.e., inorder, preorder and post order c) To display the elements in the tree.

code:

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


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

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

struct Node* insert(struct Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }

    if (data < root->data) {
        root->left = insert(root->left, data);
    } else if (data > root->data) {
        root->right = insert(root->right, data);
    }

    return root;
}

// In-order traversal (Left, Root, Right)
void inorderTraversal(struct Node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}

// Pre-order traversal (Root, Left, Right)
void preorderTraversal(struct Node* root) {
    if (root != NULL) {
```

```
        printf("%d ", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}

// Post-order traversal (Left, Right, Root)
void postorderTraversal(struct Node* root) {
    if (root != NULL) {
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf("%d ", root->data);
    }
}
void displayMenu() {
    printf("\nBinary Search Tree Operations:\n");
    printf("1. Insert a node\n");
    printf("2. In-order traversal\n");
    printf("3. Pre-order traversal\n");
    printf("4. Post-order traversal\n");
    printf("5. Exit\n");
}int main() {
    struct Node* root = NULL;  // Initialize an empty tree
    int choice, data;

    while (1) {
        displayMenu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:  // Insert a node
                printf("Enter value to insert: ");
                scanf("%d", &data);
                root = insert(root, data);
                break;

            case 2:  // In-order traversal
                printf("In-order traversal: ");
                inorderTraversal(root);
                printf("\n");
                break;

            case 3:  // Pre-order traversal
                printf("Pre-order traversal: ");
                preorderTraversal(root);
                printf("\n");
                break;

            case 4:  // Post-order traversal
```

```
            printf("Post-order traversal: ");
            postorderTraversal(root);
            printf("\n");
            break;

        case 5:  // Exit
            printf("Exiting the program.\n");
            exit(0);

        default:
            printf("Invalid choice, please try again.\n");
      }
   }

   return 0;
}
```

output:

```
5. Exit
Enter your choice: 1
Enter value to insert: 60

Binary Search Tree Operations:
1. Insert a node
2. In-order traversal
3. Pre-order traversal
4. Post-order traversal
5. Exit
Enter your choice: 2
In-order traversal: 10 20 30 40 50 60

Binary Search Tree Operations:
1. Insert a node
2. In-order traversal
3. Pre-order traversal
4. Post-order traversal
5. Exit
Enter your choice: 3
Pre-order traversal: 10 20 30 40 50 60

Binary Search Tree Operations:
1. Insert a node
2. In-order traversal
3. Pre-order traversal
4. Post-order traversal
5. Exit
Enter your choice: 4
Post-order traversal: 60 50 40 30 20 10

Binary Search Tree Operations:
1. Insert a node
2. In-order traversal
3. Pre-order traversal
4. Post-order traversal
5. Exit
Enter your choice: 5
Exiting the program.
```

9a) Write a program to traverse a graph using BFS method.

code:

```
#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:

```
PS D:\kusumaDST> cd "d:\kusumaDST\" ; if ($?) { gcc LAB9.c -o LAB9 } ; if
Enter the adjacency matrix
0 1 3 4 5 6
5 6 7 3 2 1
0 0 0 0 0 0
3 4 5 6 7 7
3 4 2 7 5 4

BFS
A -> B -> C -> D -> E ->
PS D:\kusumaDST>
```

9b) Write a program to check whether given graph is connected or not using DFS method.

code:

```
#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:

```
PS D:\kusumaDST> cd "d:\kusumaDST\" ; if ($?) { gcc LAB9b.c -o LAB9b } ;
Enter the adjacency matrix
8 9 0 6 5 8
9 0 6 4 3 6
8 9 3 4 5 6
2 4 6 7 8 2
1 3 4 5 6 0

DFS
A -> B -> D -> C -> E ->
PS D:\kusumaDST>
```

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.

code:

```
#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:

```
PS D:\kusumaDST> cd "d:\kusumaDST\" ; if ($?) { gcc LAB10.c -o LAB10 } ; if ($?)

Enter the number of employee records (N): 3

Enter the memory size (m) for the hash table: 4

Enter the four-digit key values (K) for 3 Employee Records:
0000
0001
0002

Hash Table contents are:

 T[0] --> 0
 T[1] --> 1
 T[2] --> 2
 T[3] --> -1
PS D:\kusumaDST>
```