VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

NISHANTH K S (1BM22CS183)

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
Dec 2023- March 2024
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 NISHANTH K S **(1BM22CS183)**, 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 2023-24. 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. Sneha S Bagalkot
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
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
1. **Pop**
2. **Display**

The program should print appropriate messages for stack overflow, stack underflow.

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5
int top = -1;
int stack[SIZE];

void push(int element);
int pop();
void display();

int main() {
   int choice, element;

   do {
      printf("\nStack Operations:\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 element to push: ");
            scanf("%d", &element);
            push(element);
            break;
         case 2:
            element = pop();
            if (element != -1) {
               printf("Popped element: %d\n", element);
            }
            break;
         case 3:
            display();
            break;
```

```
            case 4:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice. Please enter a valid option.\n");
        }

    } while (choice != 4);

    return 0;
}

void push(int element) {
    if (top == SIZE - 1) {
        printf("Stack Overflow. Cannot push element %d.\n", element);
    } else {
        top++;
        stack[top] = element;
        printf("Element %d pushed onto the stack.\n", element);
    }
}

int pop() {
    if (top == -1) {
        printf("Stack Underflow. Cannot pop from an empty stack.\n");
        return -1; // indicating failure
    } else {
        int element = stack[top];
        top--;
        return element;
    }
}

void display() {
    if (top == -1) {
        printf("Stack is empty.\n");
    } else {
        printf("Stack elements: ");
        for (int i = 0; i <= top; i++) {
            printf("%d ", stack[i]);
        }
        printf("\n");
    }
}
```

Output:

```
1. Push
2. Pop
3. Display stack
4. Exit
Enter choice: 1
Enter integer to be pushed: 3
Enter choice: 1
Enter integer to be pushed: 4
Enter choice: 2
Integer poppped = 4
Enter choice: 1
Enter integer to be pushed: 5
Enter choice: 3
3 5 0 0 0 0 0 0 0 0
Enter choice: 4
```

Lab program 2:

WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and

```
#include<stdio.h>
#include<ctype.h>
#define max 20
void push(char a);
char pop();
char stack[max],top =-1;
int pre(char a);

void main(){
   char infix[max],a;
   char post[max];
   printf("Enter infix expression: ");
   scanf("%s",infix);
   int j=0;
   push('(');
   for(int i=0;i<strlen(infix);i++){
```

```
        if(isalnum(infix[i])){
            post[j]=infix[i];
            j+=1;
        }
       else if((infix[i]=='+' || infix[i]=='-' || infix[i]=='/' || infix[i]=='*')){
              if(pre(infix[i])>pre(stack[top])){
                 push(infix[i]);
              }
              else if(pre(infix[i])<=pre(stack[top])){

                 while(1){
                 a=pop();
                 if(a=='('){
                    push(a);
                    break;
                 }
                 post[j]=a;
                 j+=1;
              }
              push(infix[i]);
           }


    }
}
while(top!=-1){
    char y=pop();
    if(y=='('){
        break;
       }
    post[j]=y;
    j+=1;


}
post[j]='\0';
printf("%s",post);

}

void push(char a){
    if(top>max-1){
        printf("Stack overflow");
        exit(0);
    }
    else{
```

```
        ++top;
        stack[top]=a;
    }
}

char pop(){
    if(top==-1){
        printf("Stack underflow:");
        exit(0);
    }
    else{
        return stack[top--];

    }
}

int pre(char a){
        if(a=='^'){
            return 3;
        }
        else if( a=='*' || a=='/'){
            return 2;
        }
        else if(a=='+' || a=='-'){
            return 1;
        }
        else{
            return 0;
        }
}
```

OUTPUT:

```
Enter size of expression in terms of characters: 9
Enter infix expression: a*b+c*d-e
Postfix expression: ab*cd*+e-
```

Lab program 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

b) 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

```
#include <stdio.h>
# define SIZE 100
void enqueue();
void dequeue();
void show();
int inp_arr[SIZE];
int Rear = - 1;
int Front = - 1;
main()
{
   int ch;
   while (1)
   {
      printf("1.Enqueue Operation\n");
      printf("2.Dequeue Operation\n");
      printf("3.Display the Queue\n");
      printf("4.Exit\n");
      printf("Enter your choice of operations : ");
      scanf("%d", &ch);
      switch (ch)
      {
         case 1:
         enqueue();
         break;
         case 2:
         dequeue();
         break;
         case 3:
         show();
         break;
         case 4:
         exit(0);
```

```
            default:
            printf("Incorrect choice \n");
        }
    }
}

void enqueue()
{
    int insert_item;
    if (Rear == SIZE - 1)
      printf("Overflow \n");
    else
    {
        if (Front == - 1)

        Front = 0;
        printf("Element to be inserted in the Queue\n : ");
        scanf("%d", &insert_item);
        Rear = Rear + 1;
        inp_arr[Rear] = insert_item;
    }
}

void dequeue()
{
    if (Front == - 1 || Front > Rear)
    {
        printf("Underflow \n");
        return ;
    }
    else
    {
        printf("Element deleted from the Queue: %d\n", inp_arr[Front]);
        Front = Front + 1;
    }
}

void show()
{

    if (Front == - 1)
        printf("Empty Queue \n");
    else
    {
        printf("Queue: \n");
        for (int i = Front; i <= Rear; i++)
```

```
            printf("%d ", inp_arr[i]);
        printf("\n");
    }
}
```

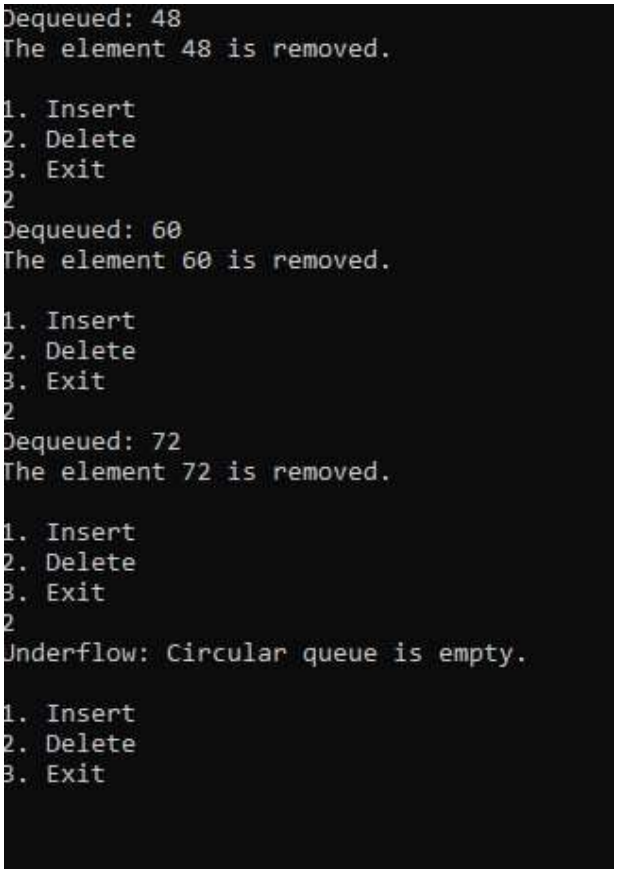
OUTPUT:

```
1. Insert
2. Delete
3. Exit
1
Enter the element to be inserted: 12
Enqueued: 12

1. Insert
2. Delete
3. Exit
1
Enter the element to be inserted: 24
Enqueued: 24

1. Insert
2. Delete
3. Exit
1
Enter the element to be inserted: 36
Enqueued: 36

1. Insert
2. Delete
3. Exit
1
Enter the element to be inserted: 48
Enqueued: 48

1. Insert
2. Delete
3. Exit
1
Enter the element to be inserted: 60
Enqueued: 60

1. Insert
2. Delete
3. Exit
1
Enter the element to be inserted: 72
Enqueued: 72

1. Insert
2. Delete
3. Exit
1
Enter the element to be inserted: 84
Overflow: Circular queue is full.

1. Insert
2. Delete
3. Exit
2
Dequeued: 12
The element 12 is removed.

1. Insert
2. Delete
3. Exit
2
Dequeued: 24
The element 24 is removed.
```

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 5
void enqueue(int element);
int dequeue();
void display();
int peep();
int front = -1;
int rear = -1;
int queue[MAX];
int main(){
do
{
   printf("\nEnter options to perform operations\n1.enqueue\n2.dequeue\n3.peep\n4.dispaly\n5.exit\n");
   int option;
   scanf("%d",&option);
   switch (option)
   {
      case 1 : printf("\nEnter the number to add : ");
               int num;
               scanf("%d",&num);
               enqueue(num);
               break;
      case 2 : printf("The deleted element is =  %d\n",dequeue());
               break;
      case 3 : printf("the front element is = %d\n",peep());
               break;
      case 4 : printf("The queue is : \n");
               display();
               break;
      case 5 : exit(0);
               break;
      default : printf("\nenter a valid number");
   }
} while (1);
}

void enqueue(int element){
   if((rear+1)%MAX==front){
      printf("queue is full");
   }else if(front==-1||rear==-1){
      front=rear=0;
      queue[rear]=element;
```

```
    }else{
        rear=(rear+1)%MAX;
        queue[rear]=element;
        printf("\nelement added\n");
    }
}
int dequeue(){
    if(front==-1||rear==-1){
        printf("\nqueue is empty\n");
        return -1;
    }else if(front==rear){
        int element = queue[front];
        front=rear=-1;
        return element;
    }else{
        int element = queue[front];
        front=(front+1)%MAX;
        return element;
    }
}
int peep(){
    if(front==-1||rear==-1){
        printf("\nque is empty\n");
        return -1;
    }else{
        return queue[front];
    }
}
void display(){
    int i;
    if(front==-1||rear==-1){
        printf("\nqueue is empty\n");
    }else{
        for(i = front ; i!=rear/*||i==rear*/ ; i=(i+1)%MAX){
            printf("%d\t",queue[i]);
        }
        printf("%d\t",queue[i]);
    }

}
```

OUTPUT:



Lab program 4:

WAP to Implement Singly Linked List with the following operations

a) Createalinkedlist.
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>

// Describes a Node
struct Node {
    int data;
```

```
    struct Node *next;
};

// Inserts an element at the start of the node
struct Node *insertstart(struct Node *start, int data) {
    struct Node *ptr = (struct Node *)malloc(sizeof(struct Node));
    ptr->data = data;
    ptr->next = start;
    return ptr;
}

// Inserts a new Node at a given index
struct Node *insertIndex(struct Node *start, int data, int index) {
    struct Node *ptr = (struct Node *)malloc(sizeof(struct Node));
    ptr->data = data;

    struct Node *p = start;
    int i = 0;
    while (i < index - 1 && p != NULL) {
        p = p->next;
        i++;
    }
    if (p == NULL) {
        printf("Index out of bounds:\n");
        free(ptr);
        return start;
    }
    ptr->next = p->next;
    p->next = ptr;
    return start;
}

// Insert at the end
struct Node *InsertEnd(struct Node *start, int data) {
    struct Node *ptr = (struct Node *)malloc(sizeof(struct Node));
    ptr->data = data;
    ptr->next = NULL;

    if (start == NULL) {
        // If the list is empty, the new node becomes the start
        return ptr;
    }

    struct Node *p = start;
    while (p->next != NULL) {
        p = p->next;
```

```
    }
    p->next = ptr;
    return start;
}

// Prints all the elements present in a Node
void display(struct Node *ptr) {
    while (ptr != NULL) {
        printf("Element: %d\n", ptr->data);
        ptr = ptr->next;
    }
}

int main() {
    struct Node *first = NULL;

    char choice;
    int newData, newIndex;

    do {
        printf("\nChoose an option:\n");
        printf("1. Insert at the beginning\n");
        printf("2. Insert at a specific index\n");
        printf("3. Insert at end\n");
        printf("4. Display the list\n");
        printf("5. Quit\n");
        printf("Enter your choice: ");
        scanf(" %c", &choice);

        switch (choice) {
            case '1':
                printf("Enter the new element to insert at the beginning: ");
                scanf("%d", &newData);
                first = insertstart(first, newData);
                break;
            case '2':
                printf("Enter the new element to insert: ");
                scanf("%d", &newData);
                printf("Enter the index to insert at: ");
                scanf("%d", &newIndex);
                first = insertIndex(first, newData, newIndex);
                break;
            case '3':
                printf("Enter the new element to insert: ");
                scanf("%d", &newData);
                first = InsertEnd(first, newData);
```

```
            break;
        case '4':
            printf("Linked List:\n");
            display(first);
            break;
        case '5':
            printf("Quitting the program.\n");
            break;
        default:
            printf("Invalid choice. Please enter a valid option.\n");
    }
  } while (choice != '5');

  return 0;
}
```

OUTPUT:

```
Choose an option:
1. Insert at the beginning
2. Insert at a specific index
3. Insert at end
4. Display the list
5. Quit
Enter your choice:
1
Enter the new element to insert at the beginning: 1

Choose an option:
1. Insert at the beginning
2. Insert at a specific index
3. Insert at end
4. Display the list
5. Quit
Enter your choice: 3
Enter the new element to insert: 2

Choose an option:
1. Insert at the beginning
2. Insert at a specific index
3. Insert at end
4. Display the list
5. Quit
Enter your choice: 2
Enter the new element to insert: 3
Enter the index to insert at: 1

Choose an option:
1. Insert at the beginning
2. Insert at a specific index
3. Insert at end
4. Display the list
5. Quit
Enter your choice: 4
Linked List:
Element: 1
Element: 3
Element: 2

Choose an option:
1. Insert at the beginning
2. Insert at a specific index
3. Insert at end
4. Display the list
5. Quit
Enter your choice: 5
Quitting the program.

Process returned 0 (0x0)   execution time : 62.938 s
Press any key to continue.
```

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.

3. **Display the contents of the linked list.**

```
#include<stdio.h>
#include<stdlib.h>
struct node {
```

```
    int data;
    struct node *next;
};
struct node *head = NULL;
void display() {
    printf("Elements are: ");
    struct node *ptr = head;
    while (ptr != NULL) {
        printf("%d -> ", ptr->data);
        ptr = ptr->next;
    }
    printf("NULL\n");
}
void insert_begin() {
    struct node *temp = (struct node*)malloc(sizeof(struct node));
    printf("Enter the value to be inserted: ");
    scanf("%d", &temp->data);
    temp->next = head;
    head = temp;
}
void delete_begin() {
    if (head == NULL) {
        printf("List is empty. Deletion not possible.\n");
        return;
    }
    struct node *temp = head;
    head = head->next;
    printf("Element deleted from the beginning: %d\n", temp->data);
    free(temp);
}
void delete_end() {
    if (head == NULL) {
        printf("List is empty. Deletion not possible.\n");
        return;
    }
    struct node *temp, *prev;
    temp = head;
    while (temp->next != NULL) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == head) {
        head = NULL;
    } else {
        prev->next = NULL;
    }
```

```
    printf("Element deleted from the end: %d\n", temp->data);
    free(temp);
}
void delete_at_position() {
    int position;
    printf("Enter the position to delete: ");
    scanf("%d", &position);

    if (head == NULL) {
        printf("List is empty. Deletion not possible.\n");
        return;
    }
    struct node *temp, *prev;
    temp = head;
    if (position == 0) {
        head = head->next;
        printf("Element at position %d deleted successfully.\n", position);
        free(temp);
        return;
    }
    for (int i = 0; temp != NULL && i < position; i++) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Position %d is out of bounds.\n", position);
        return;
    }
    prev->next = temp->next;
    printf("Element at position %d deleted successfully.\n", position);
    free(temp);
}
int main() {
    int choice;
    while (1) {
        printf("\n 1. to insert at the beginning\n 2. to delete beginning\n 3. to delete at end\n 4.
to delete at any position\n 5. to display\n 6. to exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                insert_begin();
                break;
            case 2:
                delete_begin();
                break;
```

```
            case 3:
                delete_end();
                break;
            case 4:
                delete_at_position();
                break;
            case 5:
                display();
                break;
            case 6:
                exit(0);
                break;
            default:
                printf("Enter the correct choice\n");
                break;
        }
    }
    return 0;
}
```

OUTPUT:

```
3. Exit
Choice: 1
Enter data and position: 1 0
Count: 1
Linked List: 1
Enter choice: 1
Enter data and position: 2 1
Count: 2
Linked List: 1 2
Enter choice: 1
Enter data and position: 3 2
Count: 3
Linked List: 1 2 3
Enter choice: 1
Enter data and position: 4 3
Count: 4
Linked List: 1 2 3 4
Enter choice: 2
Enter position: 0
Count: 3
Linked List: 2 3 4
Enter choice: 2
Enter position: 1
Count: 2
Linked List: 2 4
Enter choice: 2
Enter position: 1
Count: 1
Linked List: 2
```

Lab program 6:

a) WAP to Implement Single Link List with following operations: Sortthelinked

list, Reversethelinkedlist, Concatenation of two linked lists.
b) WAP to Implement Single Link List to simulate Stack & Queue Operations.

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

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

struct Node* insert(struct Node* first, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;

    if (first == NULL) {
```

```
        return newNode;
    }

    struct Node* ptr = first;
    while (ptr->next != NULL) {
        ptr = ptr->next;
    }

    ptr->next = newNode;
    return first;
}

void sort(struct Node* first) {
    if (first == NULL) {
        printf("List is empty.\n");
        return;
    }

    struct Node* ptr;
    struct Node* p;

    for (ptr = first; ptr != NULL; ptr = ptr->next) {
        bool swapped = false;
        for (p = first; p->next != NULL; p = p->next) {
            if (p->data > (p->next)->data) {
                int temp = p->data;
                p->data = (p->next)->data;
                (p->next)->data = temp;
                swapped = true;
            }
        }
        if (!swapped) {
            break;
        }
    }
    struct Node* p1 = first;
    while (p1 != NULL) {
        printf(" Element: %d", p1->data);
        p1 = p1->next;
    }
    printf("\n");
}

void display(struct Node* first) {
    struct Node* p1 = first;
    while (p1 != NULL) {
```

```
        printf(" Element: %d", p1->data);
        p1 = p1->next;
    }
    printf("\n");
}

struct Node* reverse(struct Node* first) {
    struct Node* prev = NULL;
    struct Node* current = first;
    struct Node* next = NULL;

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

    return prev;
}

struct Node* concatenate(struct Node* first1, struct Node* first2) {
    if (first1 == NULL) {
        return first2;
    }

    struct Node* ptr = first1;
    while (ptr->next != NULL) {
        ptr = ptr->next;
    }

    ptr->next = first2;
    return first1;
}

int main() {
    struct Node* first1 = NULL;
    struct Node* first2 = NULL;
    int n, ele;

    do {
        printf("\n1. Add Element to List 1\n2. Sort List 1\n3. Reverse List 1\n4. Display List
1\n");
        printf("5. Add Element to List 2\n6. Sort List 2\n7. Reverse List 2\n8. Display List
2\n");
        printf("9. Concatenate Lists\n10. Exit\n");
```

```
        scanf("%d", &n);

        switch (n) {
            case 1:
                printf("Enter the data for List 1:\n");
                scanf("%d", &ele);
                first1 = insert(first1, ele);
                break;
            case 2:
                sort(first1);
                break;
            case 3:
                first1 = reverse(first1);
                printf("List 1 reversed.\n");
                break;
            case 4:
                display(first1);
                break;
            case 5:
                printf("Enter the data for List 2:\n");
                scanf("%d", &ele);
                first2 = insert(first2, ele);
                break;
            case 6:
                sort(first2);
                break;
            case 7:
                first2 = reverse(first2);
                printf("List 2 reversed.\n");
                break;
            case 8:
                display(first2);
                break;
            case 9:
                first1 = concatenate(first1, first2);
                printf("Lists concatenated.\n");
                break;
            case 10:
                exit(0);
            default:
                printf("Enter correct choice\n");
        }
    } while (1);

    return 0;
}
```

OUTPUT:

```
1. Add Element
2. Sort
3. Reverse
4. Display
5. Exit
1
Enter the data:
1

1. Add Element
2. Sort
3. Reverse
4. Display
5. Exit
1
Enter the data:
2

1. Add Element
2. Sort
3. Reverse
4. Display
5. Exit
1
Enter the data:
3

1. Add Element
2. Sort
3. Reverse
4. Display
5. Exit
1
Enter the data:
4

1. Add Element
2. Sort
3. Reverse
4. Display
5. Exit
2
 Element: 1 Element: 2 Element: 3 Element: 4

1. Add Element
2. Sort
3. Reverse
4. Display
5. Exit
3
List reversed.

1. Add Element
2. Sort
3. Reverse
4. Display
5. Exit
4
 Element: 4 Element: 3 Element: 2 Element: 1
```

```
Original List 1: 1 -> 2 -> 3 -> NULL
Original List 2: 4 -> 5 -> 6 -> NULL
Concatenated List: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> NULL

Process returned 0 (0x0)   execution time : 0.000 s
Press any key to continue.
```

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

```
// Structure to create a node with data and the next pointer
struct node {
   int data;
   struct node * next;
};

struct node * front = NULL;
struct node * rear = NULL;

// Enqueue() operation on a queue
void enqueue(int value) {
   struct node * ptr;
   ptr = (struct node * ) malloc(sizeof(struct node));
   ptr-> data = value;
   ptr-> next = NULL;
   if ((front == NULL) && (rear == NULL)) {
      front = rear = ptr;
   } else {
      rear-> next = ptr;
      rear = ptr;
   }
   printf("Node is Inserted\n\n");
}

// Dequeue() operation on a queue
int dequeue() {
   if (front == NULL) {
      printf("\nUnderflow\n");
      return -1;
   } else {
      struct node * temp = front;
      int temp_data = front-> data;
      front = front-> next;
      free(temp);
      return temp_data;
   }
}

// Display all elements of the queue
void display() {
   struct node * temp;
   if ((front == NULL) && (rear == NULL)) {
      printf("\nQueue is Empty\n");
   } else {
      printf("The queue is \n");
```

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

int main() {
    int choice, value;
    printf("\nImplementation of Queue using Linked List\n");
    while (choice != 4) {
        printf("1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");
        printf("\nEnter your choice : ");
        scanf("%d", & choice);
        switch (choice) {
            case 1:
                printf("\nEnter the value to insert: ");
                scanf("%d", & value);
                enqueue(value);
                break;
            case 2:
                printf("Popped element is :%d\n", dequeue());
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);
                break;
            default:
                printf("\nWrong Choice\n");
        }
    }
    return 0;
}
```

```
Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the element to enqueue: 1

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the element to enqueue: 2

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the element to enqueue: 3

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the element to enqueue: 4

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 2 3 4

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 3 4
```

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

struct node {
   int info;
   struct node *ptr;
}*top,*top1,*temp;

int count = 0;
void push(int data) {
    if (top == NULL)
   {
      top =(struct node *)malloc(1*sizeof(struct node));
      top->ptr = NULL;
      top->info = data;
   }
   else
   {
      temp =(struct node *)malloc(1*sizeof(struct node));
      temp->ptr = top;
      temp->info = data;
      top = temp;
   }
   count++;
   printf("Node is Inserted\n\n");
}

int pop() {
    top1 = top;

   if (top1 == NULL)
   {
      printf("\nStack Underflow\n");
      return -1;
   }
   else
      top1 = top1->ptr;
   int popped = top->info;
   free(top);
   top = top1;
   count--;
   return popped;
}

void display() {
```

```
    // Display the elements of the stack
    top1 = top;

    if (top1 == NULL)
    {
        printf("\nStack Underflow\n");
        return;
    }

    printf("The stack is \n");
    while (top1 != NULL)
    {
        printf("%d--->", top1->info);
        top1 = top1->ptr;
    }
    printf("NULL\n\n");

}

int main() {
    int choice, value;
    printf("\nImplementation of Stack using Linked List\n");
    while (1) {
        printf("\n1. Push\n2. Pop\n3. Display\n4. Exit\n");
        printf("\nEnter your choice : ");
        scanf("%d", &choice);
        switch (choice) {
        case 1:
            printf("\nEnter the value to insert: ");
            scanf("%d", &value);
            push(value);
            break;
        case 2:
            printf("Popped element is :%d\n", pop());
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
            break;
        default:
            printf("\nWrong Choice\n");
        }
    }
}
```

```
Stack Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the element to push: 1

Stack Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the element to push: 2

Stack Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the element to push: 3

Stack Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the element to push: 4

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

Stack Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack elements: 3 2 1

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

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

Lab Program 7:
WAP to Implement doubly link list with primitive operations

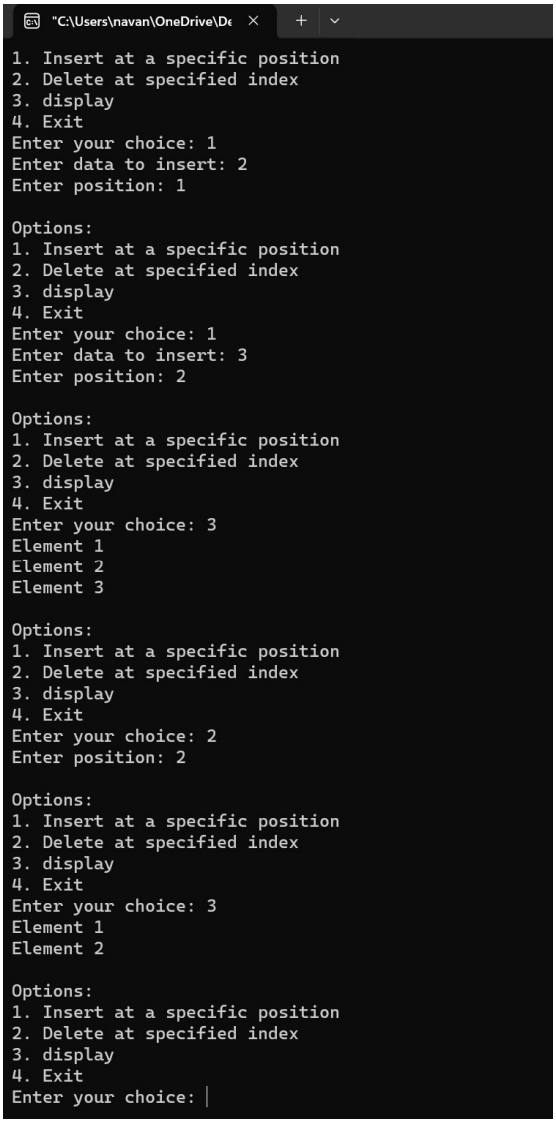
a) Create a doubly linked list.
b) Insert a new node to the left of the node.
c) Delete the node based on a specific value
d) Display the contents of the list

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


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

struct Node* head = NULL;

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

void insertNodeToLeft(struct Node* node, int newData) {
   if (node == NULL) {
      printf("Error: Cannot insert to the left of a NULL node.\n");
      return;
   }

   struct Node* newNode = createNode(newData);

   newNode->next = node;
   newNode->prev = node->prev;

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

```
    node->prev = newNode;
}

void deleteNodeByValue(int value) {
    struct Node* current = head;

    while (current != NULL && current->data != value) {
        current = current->next;
    }

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

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

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

    free(current);
}

void displayList() {
    struct Node* current = head;

    printf("Doubly Linked List: ");
    while (current != NULL) {
        printf("%d <-> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}

int main() {
    head = createNode(1);
    struct Node* second = createNode(2);
    struct Node* third = createNode(3);

    head->next = second;
```

```
    second->prev = head;
    second->next = third;
    third->prev = second;

    insertNodeToLeft(second, 5);

    deleteNodeByValue(2);

    displayList();

    return 0;
}
```

OUTPUT:



Lab program 8:

Write a program

a) ToconstructabinarySearchtree.
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>
struct node
{
   int data;
   struct node *left;
   struct node *right;
};

struct node *create(int val)
{
   struct node *ptr = (struct node*)malloc(sizeof(struct node));
   ptr->data=val;
   ptr->left=ptr->right=NULL;
   return ptr;
}
struct node *insert(struct node *root,int val)
{
   if(root == NULL)
   {
      return create(val);
   }
   if(val < root->data)
   {
      root->left = insert(root->left,val);
   }else if(val > root->data)
   {
      return insert(root->right,val);
   }
   return root;
}
void preOrder(struct node *root)
{
   if(root != NULL)
   {
      printf("%d",root->data);
      preOrder(root->left);
      preOrder(root->right);
```

```
    }
}
void inOrder(struct node* root)
{
    if (root != NULL)
        {
        inOrder(root->left);
        printf("%d ", root->data);
        inOrder(root->right);
    }
}
void postOrder(struct node* root)
 {
    if (root != NULL)
        {
        postOrder(root->left);
        postOrder(root->right);
        printf("%d ", root->data);
    }
}
void display(struct node* root) {
    printf("Elements in the tree: ");
    inOrder(root);
    printf("\n");
}
int main() {
    struct node* root = NULL;
    int choice, value;

    do {
        printf("\nBinary Search Tree Menu:\n");
        printf("1. Insert element\n");
        printf("2. Display elements\n");
        printf("3. In-order traversal\n");
        printf("4. Pre-order traversal\n");
        printf("5. Post-order traversal\n");
        printf("0. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the element to insert: ");
                scanf("%d", &value);
                root = insert(root, value);
                break;
```

```
            case 2:
                display(root);
                break;
            case 3:
                printf("In-order traversal: ");
                inOrder(root);
                printf("\n");
                break;
            case 4:
                printf("Pre-order traversal: ");
                preOrder(root);
                printf("\n");
                break;
            case 5:
                printf("Post-order traversal: ");
                postOrder(root);
                printf("\n");
                break;
            case 0:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 0);

    return 0;
}
```

OUTPUT:

```
Enter your choice:
1. In-order traversal
2. Pre-order traversal
3. Post-order traversal
2
Elements in the tree (pre-order traversal): 50 30 20 40 70 60 80
```

```
Enter your choice:
1. In-order traversal
2. Pre-order traversal
3. Post-order traversal
3
Elements in the tree (post-order traversal): 20 40 30 60 80 70 50
```

```
Enter your choice:
1. In-order traversal
2. Pre-order traversal
3. Post-order traversal
1
Elements in the tree (in-order traversal): 20 30 40 50 60 70 80
```

Lab program 9:

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

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

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

#define MAX_NODES 100

int adjMatrix[MAX_NODES][MAX_NODES];
int visited[MAX_NODES];

void DFS(int node, int nodes) {
    visited[node] = 1;
    for (int i = 0; i < nodes; i++) {
        if (adjMatrix[node][i] && !visited[i]) {
            DFS(i, nodes);
        }
    }
}

int isConnected(int nodes) {
    for (int i = 0; i < nodes; i++) {
        visited[i] = 0;
    }

    DFS(0, nodes);

    for (int i = 0; i < nodes; i++) {
        if (!visited[i]) {
            return 0;
        }
    }
    return 1;
}

int main() {
    int nodes;

    printf("Enter the number of nodes: ");
    scanf("%d", &nodes);
```

```
    printf("Enter adjacency matrix for the graph:\n");
    for (int i = 0; i < nodes; i++) {
        for (int j = 0; j < nodes; j++) {
            scanf("%d", &adjMatrix[i][j]);
        }
    }

    if (isConnected(nodes)) {
        printf("The graph is connected.\n");
    } else {
        printf("The graph is not connected.\n");
    }

    return 0;
}
```

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

#define MAX_NODES 100

struct Queue {
    int items[MAX_NODES];
    int front;
    int rear;
};

struct Queue* createQueue() {
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
    queue->front = -1;
    queue->rear = -1;
    return queue;
}

int isEmpty(struct Queue* queue) {
    if (queue->rear == -1)
        return 1;
    else
        return 0;
}

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

```
    if (queue->rear == MAX_NODES - 1)
        printf("\nQueue is Full!!");
    else {
        if (queue->front == -1)
            queue->front = 0;
        queue->rear++;
        queue->items[queue->rear] = value;
    }
}

int dequeue(struct Queue* queue) {
    int item;
    if (isEmpty(queue)) {
        printf("\nQueue is Empty!!");
        item = -1;
    } else {
        item = queue->items[queue->front];
        queue->front++;
        if (queue->front > queue->rear) {
            queue->front = queue->rear = -1;
        }
    }
    return item;
}

int adjMatrix[MAX_NODES][MAX_NODES];
int visited[MAX_NODES];

void BFS(int start, int nodes) {
    struct Queue* queue = createQueue();

    visited[start] = 1;
    enqueue(queue, start);

    printf("Breadth First Search starting from node %d: ", start);

    while (!isEmpty(queue)) {
        int currentNode = dequeue(queue);
        printf("%d ", currentNode);

        for (int i = 0; i < nodes; i++) {
            if (adjMatrix[currentNode][i] && !visited[i]) {
                visited[i] = 1;
                enqueue(queue, i);
            }
        }
```

```
    }
    printf("\n");
}

int main() {
    int nodes, startNode;

    printf("Enter the number of nodes: ");
    scanf("%d", &nodes);

    printf("Enter adjacency matrix for the graph:\n");
    for (int i = 0; i < nodes; i++) {
        for (int j = 0; j < nodes; j++) {
            scanf("%d", &adjMatrix[i][j]);
        }
    }

    for (int i = 0; i < nodes; i++) {
        visited[i] = 0;
    }

    printf("Enter the starting node for BFS: ");
    scanf("%d", &startNode);

    BFS(startNode, nodes);

    return 0;
}
```

OUTPUT:

```
Enter the number of nodes: 5
Enter adjacency matrix for the graph:
0 1 1 0 1
1 0 1 1 0
1 1 0 1 0
0 1 1 0 1
1 0 0 1 0
Enter the starting node for BFS: 3
```

```
Enter the number of nodes: 5
Enter adjacency matrix for the graph:
0 1 1 0 1
1 0 1 1 0
1 1 0 1 0
0 1 1 0 1
1 0 0 1 0
The graph is connected.

Process returned 0 (0x0)   execution time : 134.374 s
Press any key to continue.
```

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>
#include <string.h>

#define TABLE_SIZE 100
#define KEY_LENGTH 5
#define MAX_NAME_LENGTH 50
#define MAX_DESIGNATION_LENGTH 50

struct Employee {
   char key[KEY_LENGTH];
   char name[MAX_NAME_LENGTH];
   char designation[MAX_DESIGNATION_LENGTH];
   float salary;
};

struct HashTable {
   struct Employee* table[TABLE_SIZE];
};

int hash_function(const char* key, int m) {
   int sum = 0;
   for (int i = 0; key[i] != '\0'; i++) {
      sum += key[i];
   }
   return sum % m;
}

void insert(struct HashTable* ht, struct Employee* emp) {
   int index = hash_function(emp->key, TABLE_SIZE);

   while (ht->table[index] != NULL) {
      index = (index + 1) % TABLE_SIZE;
```

```
    }
    ht->table[index] = emp;
}

struct Employee* search(struct HashTable* ht, const char* key) {
    int index = hash_function(key, TABLE_SIZE);

    while (ht->table[index] != NULL) {
        if (strcmp(ht->table[index]->key, key) == 0) {
            return ht->table[index];
        }
        index = (index + 1) % TABLE_SIZE;
    }
    return NULL;
}

int main() {
    struct HashTable ht;
    struct Employee* emp;
    char key[KEY_LENGTH];
    char filename[100];
    char line[100];
    FILE* file;

    for (int i = 0; i < TABLE_SIZE; i++) {
        ht.table[i] = NULL;
    }

    printf("Enter the filename containing employee records: ");
    scanf("%s", filename);

    file = fopen(filename, "r");
    if (file == NULL) {
        printf("Error opening file.\n");
        return 1;
    }

    while (fgets(line, sizeof(line), file)) {
        emp = (struct Employee*)malloc(sizeof(struct Employee));
        sscanf(line, "%s %s %s %f", emp->key, emp->name, emp->designation, &emp-
>salary);
        insert(&ht, emp);
    }

    fclose(file);
```

```
    int choice;
    do {
        printf("\n1. Search Employee\n");
        printf("2. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the key to search: ");
                scanf("%s", key);
                emp = search(&ht, key);
                if (emp != NULL) {
                    printf("Employee record found with key %s:\n", emp->key);
                    printf("Name: %s\n", emp->name);
                    printf("Designation: %s\n", emp->designation);
                    printf("Salary: %.2f\n", emp->salary);
                } else {
                    printf("Employee record not 7found for key %s\n", key);
                }
                break;
            case 2:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Please enter again.\n");
        }
    } while (choice != 2);

    for (int i = 0; i < TABLE_SIZE; i++) {
        if (ht.table[i] != NULL) {
            free(ht.table[i]);
        }
    }

    return 0;
}
```

Output:

```
Enter the filename containing employee records: Employee.txt

1. Search Employee
2. Exit
Enter your choice: 1
Enter the key to search: 1
Employee record found with key 1:
Name: Navanith
Designation: CEO
Salary: 10000.00

1. Search Employee
2. Exit
Enter your choice: 1
Enter the key to search: 2
Employee record found with key 2:
Name: Nishanth
Designation: CEO
Salary: 10000.00

1. Search Employee
2. Exit
Enter your choice: 1
Enter the key to search: 3
Employee record found with key 3:
Name: Pranav
Designation: CEO
Salary: 10000.00

1. Search Employee
2. Exit
Enter your choice: 1
Enter the key to search: 4
Employee record found with key 4:
Name: NithinCEO
Designation: 10000
Salary: 2154132992.00

1. Search Employee
2. Exit
Enter your choice: 1
Enter the key to search: 5
Employee record found with key 5:
Name: Navneeth
Designation: CEO
Salary: 10000.00

1. Search Employee
2. Exit
Enter your choice: 2
Exiting...

Process returned 0 (0x0)   execution time : 27.868 s
Press any key to continue.
```

LEET CODE:

1. **leetCode/155MinStack.c**

```
#include <stdlib.h>
typedef struct {
    int* stack;
    int top;
    int min;
    int size;
} MinStack;

MinStack* minStackCreate() {
    MinStack* obj = (MinStack*)malloc(sizeof(MinStack));
    obj->stack = (int*)malloc(sizeof(int) * 10); // initial size of 10
    obj->top = -1;
    obj->min = INT_MAX;
    obj->size = 10;
    return obj;
}

void minStackPush(MinStack* obj, int val) {
    if (obj->top == obj->size - 1) {
        obj->size *= 2;
        obj->stack = (int*)realloc(obj->stack, sizeof(int) * obj->size);
    }
    obj->stack[++obj->top] = val;
    obj->min = val < obj->min ? val : obj->min;
}

void minStackPop(MinStack* obj) {
    if (obj->top == -1)
        return;
    if (obj->stack[obj->top--] == obj->min) {
        int min = INT_MAX;
        for (int i = 0; i <= obj->top; i++) {
            min = obj->stack[i] < min ? obj->stack[i] : min;
        }
        obj->min = min;
    }
}

int minStackTop(MinStack* obj) {
    if (obj->top == -1)
        return -1;
    return obj->stack[obj->top];
```

```
}

int minStackGetMin(MinStack* obj) { return obj->min; }

void minStackFree(MinStack* obj) {
    free(obj->stack);
    free(obj);
}
```

2. **leetCode/61RotateList**

```
struct ListNode* rotateRight(struct ListNode* head, int k) {
    if (head == NULL || k == 0) return head;
    struct ListNode* n = head;
    int i=0;
    while(n!=NULL){
        i++;
        n = n->next;
    }
    struct ListNode* temp1;
    struct ListNode* temp2;
    int turns = k%i;
    for (int i = 0; i < turns; i++) {
        temp2 = head;
        while (temp2->next != NULL) {
            temp1 = temp2;
            temp2 = temp2->next;
        }
        temp2->next = head;
        head = temp2;
        temp1->next = NULL;
    }
    return head;
}
```

3. **leetCode/725SplitLinkedListinParts.c**

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
/**
```

```
 * Note: The returned array must be malloced, assume caller calls free().
 */
typedef struct ListNode lnode;
int get_len(lnode* head) {
    int n = 0;
    while (head) {
        n++;
        head = head->next;
    }
    return n;
}
struct ListNode** splitListToParts(struct ListNode* head, int k,
                                   int* returnSize) {
    int n = get_len(head), elems, i, j;
    *returnSize = k;
    lnode **list = (lnode**)calloc(k, sizeof(lnode*)), *t = head;
    if (n > k) {
        for (i = 0; i < k; i++) {
            elems = i < n % k ? n / k + 1 : n / k;
            j = 0;
            list[i] = head;
            t = head;
            while (j++ < elems) {
                t = head;
                head = head->next;
            }
            t->next = NULL;
        }
    } else {
        for (i = 0; i < n; i++) {
            list[i] = head;
            head = head->next;
            list[i]->next = NULL;
        }
    }
    return list;
}
```

4. **leetCode/725SplitLinkedListinParts.c**

```
 struct ListNode {
    int val;
    struct ListNode *next;
```

```
};

struct ListNode* reverseBetween(struct ListNode* head, int left, int right){

    struct ListNode *p = head,*k=head;
    int i = 1,kk=0;
    int a[100000];
    while(p!=0){
        if (i>=left && i<=right){
            a[kk++]=p->val;
        }
        p=p->next;
        i++;
    }
    i=1;
    while(k!=0){
        if (i>=left && i<=right){
            k->val=a[--kk];
        }
        k=k->next;
        i++;
    }
    return head;
}
```