**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**

****

**LAB REPORT**

**On**

**DATA STRUCTURES (23CS3PCDST)**

**Submitted by**

**RAYHAN SADAT (1BM23CS264)**

**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 RAYHAN SADAT KANEKAL**(1BM23CS264)**, 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.

**Prof. Namratha M** **Dr. Kavitha Sooda**

Assistant Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

**Index Sheet**

|  |  |  |
| --- | --- | --- |
| **Sl.**  **No.** | **Experiment Title** | **Page No.** |
| 1 | Working of stack | 4-6 |
| 2 | Infix to postfix | 7-9 |
| 3 | Working of queue | 10-14 |
| 4 | Singly linked list -insertion | 15-19 |
| 5 | Singly linked list-deletion | 20-25 |
| 6 | sort ,reverse and concatenate;implementing stack and queue | 26-36 |
| 7 | Doubly linked list | 37-41 |
| 8 | Binary | 42-43 |
| 9 | BFS traversal;DFS traversal | 44-45 |
| 10 | Linear probing | 46-49 |

**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. |

**Leetcode Program 1:**

**Move Zeroes**

void moveZeroes(int\* nums, int numsSize) {

int \*ptr=nums,temp,i,\*j;

for (i=0;i<numsSize;i++)

{

if (\*ptr !=0)

ptr++;

else

{

j=ptr;

while (j<&nums[numsSize-1])

{

temp=\*j;

\*j=\*(j+1);

\*(j+1)=temp;

j++;

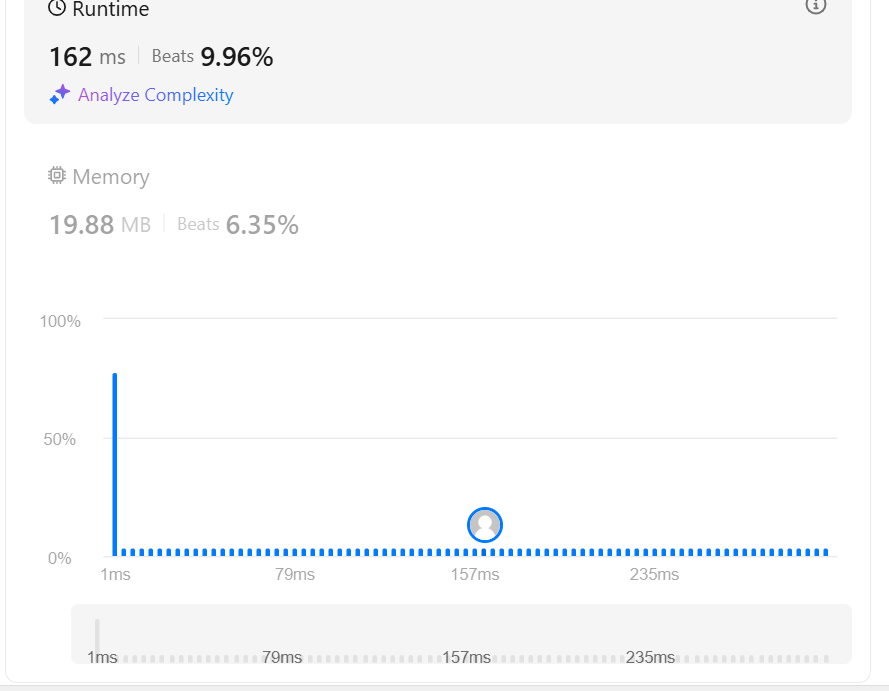
}

}

}

}

**Output:**

****

**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:**

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A computer screen with white text

Description automatically generated

**Lab program 2:**

**Write a Program 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>

#include <ctype.h>

char infix[100], stack[100];

int top = -1;

void push(char a)

{

if (top == 99)

{

printf("Overflow\n");

}

else

{

top++;

stack[top] = a;

}

}

char pop()

{

if (top == -1)

{

printf("Underflow\n");

return '\0';

}

return stack[top--];

}

int priority(char b) {

if (b == '(') return 0;

if (b == '\*' || b == '/') return 2;

if (b == '^') return 3;

if (b == '+' || b == '-') return 1;

return -1;

}

int main() {

int i;

printf("Enter your expression: ");

scanf("%s", infix);

int length = strlen(infix);

infix[length] = ')';

infix[length + 1] = '\0';

push('(');

length = strlen(infix);

for (i = 0; i < length; i++)

{

if (infix[i] == '(')

{

push(infix[i]);

}

else if (isalnum(infix[i]))

{

printf("%c", infix[i]); // Print operands directly

}

else if (infix[i] == '^' || infix[i] == '-' || infix[i] == '+' || infix[i] == '\*' || infix[i] == '/')

{

while (priority(stack[top]) >= priority(infix[i]))

{

printf("%c", pop());

}

push(infix[i]);

}

else if (infix[i] == ')')

{

while (stack[top] != '(')

{

printf("%c", pop());

}

pop();

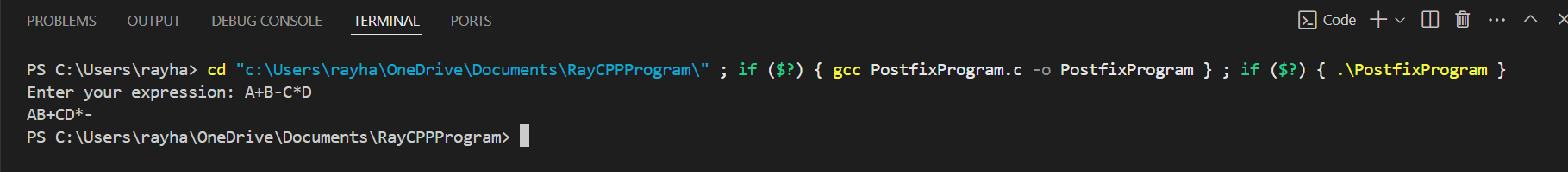
}

}

return 0;

}

**OUTPUT:**

****

**Leetcode Program 2:**

**Majority Element**

int majorityElement(int\* nums, int numsSize) {

int prev\_value = nums[0];

int count = 1;

for (int i = 1; i < numsSize; i++) {

if (nums[i] == prev\_value) {

count += 1;

} else {

count = count - 1;

if (count == 0) {

prev\_value = nums[i];

count = 1;

}

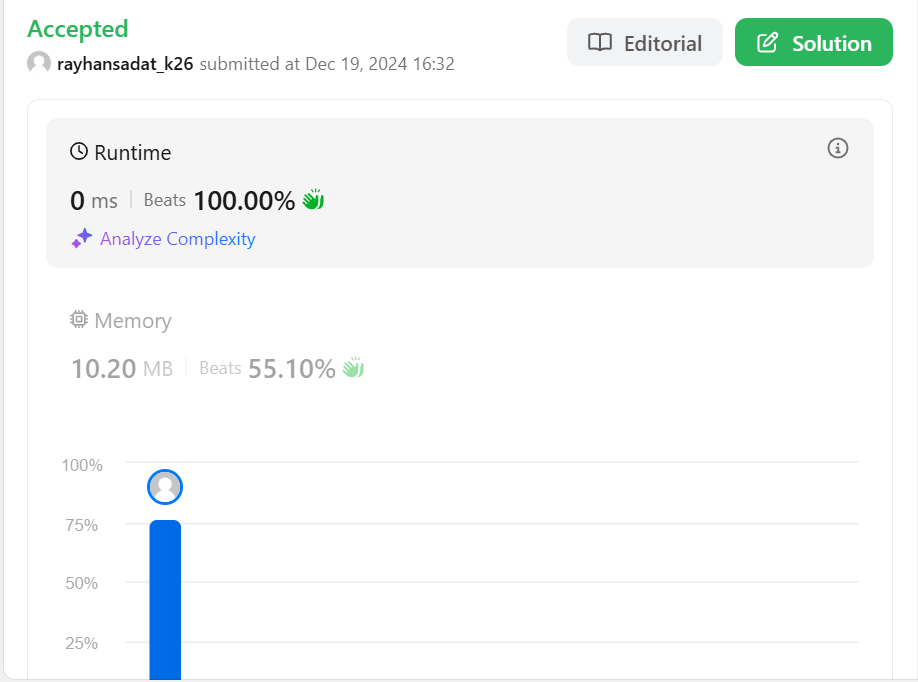
}

}

return prev\_value;

}

**OUTPUT:**

****

**Lab Program 3:**

**WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert,  
Delete, Display The program should print appropriate messages for queue empty and queue overflow conditions**

#include <stdio.h>

int Size=6;

int front=-1;

int rear=-1;

int Q[1000];

void reset()

{

if (front==rear)

{

front=rear=-1;

}

}

int insert(int a)

{

if(front==-1 && rear==-1)

{

front+=1;

rear+=1;

Q[rear]=a;

printf("Element instered is %d\n",a);

}

else if(rear==Size-1)

{

printf("Queue is full\n");

}

else

{

rear+=1;

Q[rear]=a;

printf("Element inserted is %d\n",a);

}

}

void delete()

{

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

{

printf("Underflow\n");

}

else

{

printf("The element deleted in the Queue is %d\n",Q[front]);

front+=1;

reset();

}

}

void display()

{

int i;

if ((front==rear)&&(front!=0))

{

printf("No elements in Queue\n");

}

else

{

for (i=0;i<Size;i++)

{

printf("%d\n",Q[i]);

}

}

}

void main()

{

int var,ins;

for(int i=0;i<100;i++)

{

printf("Enter 1 to insert and 2 to delete and 3 to display,4 to exit\n");

scanf("%d",&var);

if (var==1)

{

printf("Enter element to insert");

scanf("%d",&ins);

insert(ins);

}

else if (var==2)

{

delete();

}

else if (var==3)

{

display();

}

else

{

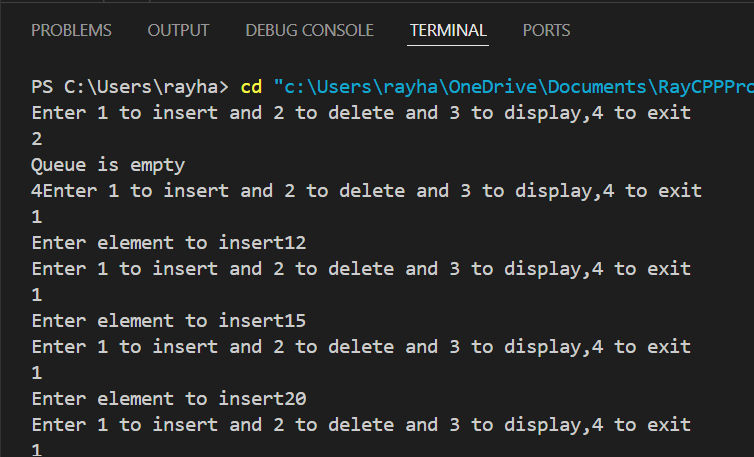
printf("Invalid input\n");

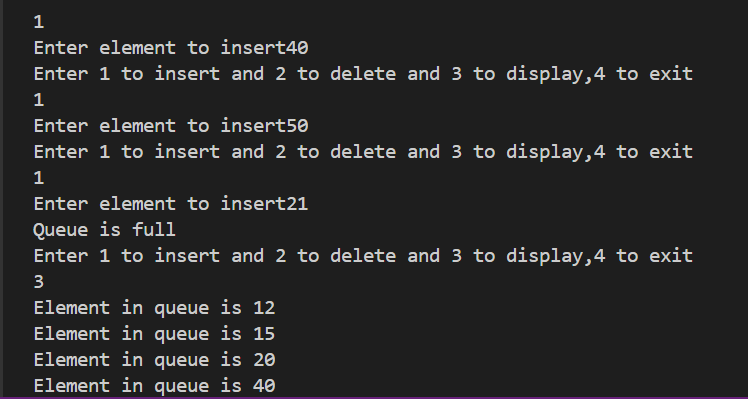
}

}

}

**OUTPUT:**

****

****

**LAB PROGRAM 4**

**WAP to simulate the working of a circular queue of integers using an array. Provide the following operations:  
Insert, Delete &amp; Display The program should print appropriate messages for queue empty and queue overflow conditions**

#include <stdio.h>

int front=-1;

int rear=-1;

int Size=4;

int CircularQ[4];

void reset()

{

if (front==rear);

front=rear=-1;

}

int enque(int a)

{

if (front==-1 && rear==-1)

{

front+=1;

rear+=1;

CircularQ[rear]=a;

printf("The number inserted is: %d\n",a);

}

else if (rear==Size-1 && front!=0)

{

rear=0;

CircularQ[rear]=a;

printf("The number inserted is: %d\n",a);

}

else if ((rear==Size-1 && front==0)||(rear+1%Size==front))

{

printf("Circular Queue is full\n");

}

else

{

rear+=1;

CircularQ[rear]=a;

printf("The element inserted is: %d\n",a);

}

}

int deque()

{

if (front==-1)

{

printf("Underflow\n");

}

else if (front==Size-1)

{

printf("The number deleted is %d\n",CircularQ[front]);

front=-1;

void reset();

}

else

{

printf("The number deleted is %d\n",CircularQ[front]);

front+=1;

void reset();

}

}

void display()

{

if (front==-1)

{

printf("No elements in Circular Queue\n");

}

else if (front>rear)

{

for (int j=0;j<=rear;j++)

{

printf("%d\n",CircularQ[j]);

}

for (int i=front;i<Size;i++)

{

printf("%d\n",CircularQ[i]);

}

}

else if (rear >= front)

{

for (int i = front; i <= rear; i++)

printf("%d\n",CircularQ[i]);

}

}

void main()

{

int option,ins;

do

{

printf("Enter 1 to insert and 2 to delete and 3 to display,4 to exit\n");

scanf("%d",&option);

if (option==1)

{

printf("Enter element to insert");

scanf("%d",&ins);

enque(ins);

}

else if (option==2)

{

deque();

}

else if (option==3)

{

display();

}

else if (option==4)

{

exit(0);

}

else

{

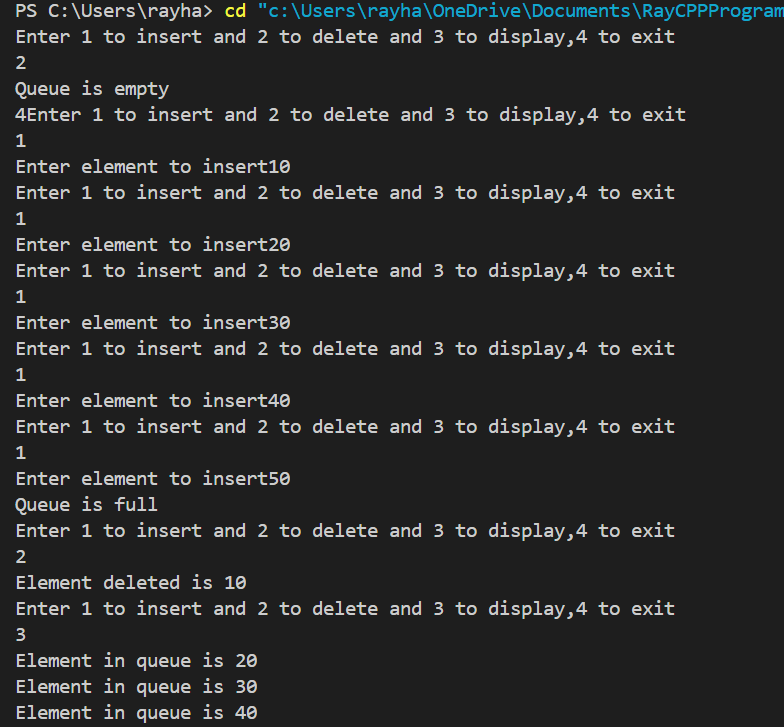
printf("Invalid input\n");

}

}while(option!=4);

}

**Output:**

****

**Lab program 5:**

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

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node\* head=NULL;

struct node\* ptr=NULL;

struct node\* createnode(int value)

{

struct node\* newnode=(struct node\*)malloc(sizeof(struct node));

newnode->data=value;

newnode->next=NULL;

}

void createll(int n)

{

if (n==0)

{

printf("Empty list");

exit(0);

}

int val;

for (int i=0;i<n;i++)

{

printf("Enter value to enter in new node");

scanf("%d",&val);

struct node\* newnode=createnode(val);

if (head==NULL)

{

head=newnode;

}

else

{

ptr->next=newnode;

}

ptr=newnode;

}

}

void printll()

{

ptr=head;

while(ptr->next!=NULL)

{

printf("%d->",ptr->data);

ptr=ptr->next;

}

printf("%d\nEND OF LINKED LIST\n",ptr->data);

}

void insert\_at\_beg(int value)

{

struct node\* newnode = createnode(value);

struct node\* insertptr;

insertptr=head;

head=newnode;

head->next=insertptr;

}

void insert\_at\_end(int value)

{

struct node\* newnode = createnode(value);

ptr->next=newnode;

ptr=newnode;

}

void delete\_at\_beg()

{

if (head==NULL)

{

printf("List is empty nothing to delete");

exit(0);

}

struct node\* deleteptr;

deleteptr=head;

head=head->next;

printf("Element deleted is %d\n",deleteptr->data);

free(deleteptr);

}

void delete\_at\_end()

{

if (head==NULL)

{

printf("List is empty nothing to delete");

exit(0);

}

struct node\* deleteptr;

struct node\* preptr;

preptr=head;

while (preptr->next!=NULL)

{

deleteptr=preptr;

preptr=preptr->next;

}

deleteptr->next=NULL;

ptr=deleteptr;

printf("Element deleted is %d\n",preptr->data);

free(preptr);

}

void deletell()

{

struct node\* preptr;

preptr=head;

while (preptr->next!=NULL)

{

delete\_at\_beg();

}

}

void main()

{

createll(5);

insert\_at\_beg(0);

insert\_at\_end(60);

printll();

delete\_at\_beg();

delete\_at\_end();

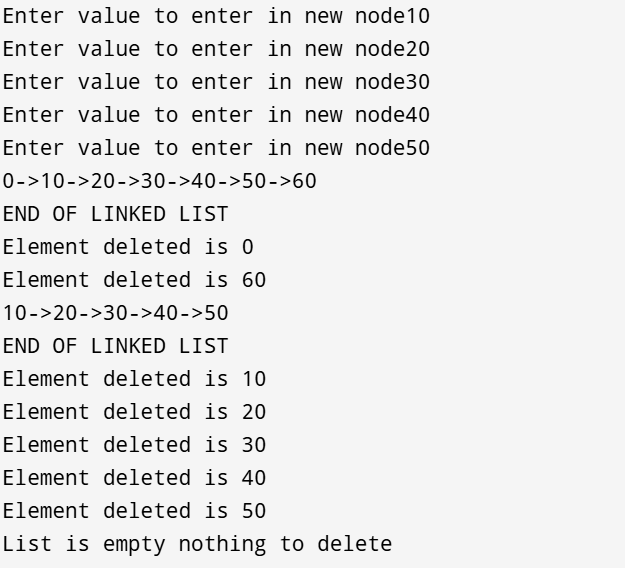
printll();

deletell();

printll();

}

**OUTPUT:**

****

**LAB PROGRAM 6:**

**6a) WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists**

int choice, data;

struct Node\* temp;

while (1) {

printf("\nLinked List Operations:\n");

printf("1. Insert a node at the end\n");

printf("2. Sort the linked list\n");

printf("3. Concatenate two linked lists\n");

printf("4. Reverse the linked list\n");

printf("5. Print the linked list\n");

printf("6. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to insert at the end: ");

scanf("%d", &data);

temp = createNode(data);

if (head1 == NULL) {

head1 = temp;

} else {

struct Node\* tempHead = head1;

while (tempHead->next != NULL) {

tempHead = tempHead->next;

}

tempHead->next = temp;

}

break;

case 2:

sortList(head1);

printf("Linked list sorted.\n");

break;

case 3:

printf("Enter data for second linked list:\n");

int more = 1;

while (more) {

printf("Enter data: ");

scanf("%d", &data);

struct Node\* temp2 = createNode(data);

if (head2 == NULL) {

head2 = temp2;

} else {

struct Node\* tempHead2 = head2;

while (tempHead2->next != NULL) {

tempHead2 = tempHead2->next;

}

tempHead2->next = temp2;

}

printf("Enter 1 to add more nodes to the second list, 0 to stop: ");

scanf("%d", &more);

}

concatenateLists(&head1, head2);

printf("Linked lists concatenated.\n");

break;

case 4:

reverseList(&head1);

printf("Linked list reversed.\n");

break;

case 5:

printf("Linked List: ");

printList(head1);

break;

case 6:

printf("Exiting...\n");

exit(0);

break;

default:

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

}

}

return 0;

}

**OUPUT:**

Linked List Operations:

1. Insert a node at the end

2. Sort the linked list

3. Concatenate two linked lists

4. Reverse the linked list

5. Print the linked list

6. Exit

Enter your choice: 1

Enter data to insert at the end: 20

Linked List Operations:

1. Insert a node at the end

2. Sort the linked list

3. Concatenate two linked lists

4. Reverse the linked list

5. Print the linked list

6. Exit

Enter your choice: 1

Enter data to insert at the end: 30

Linked List Operations:

1. Insert a node at the end

2. Sort the linked list

3. Concatenate two linked lists

4. Reverse the linked list

5. Print the linked list

6. Exit

Enter your choice: 1

Enter data to insert at the end: 40

Linked List Operations:

1. Insert a node at the end

2. Sort the linked list

3. Concatenate two linked lists

4. Reverse the linked list

5. Print the linked list

6. Exit

Enter your choice: 1

Enter data to insert at the end: 10

Linked List Operations:

1. Insert a node at the end

2. Sort the linked list

3. Concatenate two linked lists

4. Reverse the linked list

5. Print the linked list

6. Exit

Enter your choice: 2

Linked list sorted.

Linked List Operations:

1. Insert a node at the end

2. Sort the linked list

3. Concatenate two linked lists

4. Reverse the linked list

5. Print the linked list

6. Exit

Enter your choice: 5

Linked List: 10 -> 20 -> 30 -> 40 -> NULL

Linked List Operations:

1. Insert a node at the end

2. Sort the linked list

3. Concatenate two linked lists

4. Reverse the linked list

5. Print the linked list

6. Exit

Enter your choice: 3

Enter data for second linked list:

Enter data: 12

Enter 1 to add more nodes to the second list, 0 to stop: 1

Enter data: 24

Enter 1 to add more nodes to the second list, 0 to stop: 0

Linked lists concatenated.

Linked List Operations:

1. Insert a node at the end

2. Sort the linked list

3. Concatenate two linked lists

4. Reverse the linked list

5. Print the linked list

6. Exit

Enter your choice: 5

Linked List: 10 -> 20 -> 30 -> 40 -> 12 -> 24 -> NULL

**6b) WAP to create a stack using a linked list:**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node\* head=NULL;

struct node\* ptr=NULL;

int top1=-1;

struct node\* createnode(int value)

{

struct node\* newnode=(struct node\*)malloc(sizeof(struct node));

newnode->data=value;

newnode->next=NULL;

}

void printstack()

{

ptr=head;

while(ptr->next!=NULL)

{

printf("%d->",ptr->data);

ptr=ptr->next;

}

printf("%d\nEND OF STACK\n",ptr->data);

}

void push(int value)

{

if (top1==4)

{

printf("Stack is full, element not pushed\n");

}

else

{

struct node\* newnode = createnode(value);

if (head==NULL)

{

head=newnode;

ptr=newnode;

}

else

{

ptr->next=newnode;

ptr=newnode;

}

printf("Element pushed is %d\n",ptr->data);

top1++;

}

}

void pop()

{

if (head == NULL)

{

printf("Stack is empty, nothing to delete\n");

}

else if (head->next == NULL)

{

printf("Element deleted is %d\n", head->data);

free(head);

head = NULL;

ptr = NULL;

top1--;

}

else

{

struct node\* deleteptr;

struct node\* preptr = head;

while (preptr->next->next != NULL)

{

preptr = preptr->next;

}

deleteptr = preptr->next;

preptr->next = NULL;

ptr = preptr;

printf("Element deleted is %d\n", deleteptr->data);

free(deleteptr);

top1--;

}

}

void peek()

{

if (ptr==NULL)

{

printf("Stack is empty \n");

}

else

{

printf("top is %d\n",ptr->data);

}

}

void main()

{

int n,pushedValue;

for(int i=0;i<=10000;i++)

{

printf("Enter 1 to push,2 to pop 3 to display,4 to peek and 5 to exit");

scanf("%d",&n);

if (n==1)

{

printf("Enter value to be pushed");

scanf("%d",&pushedValue);

push(pushedValue);

}

else if (n==2)

{

pop();

}

else if(n==3)

{

printstack();

}

else if (n==4)

{

peek();

}

else if(n==5)

{

exit(0);

}

else

{

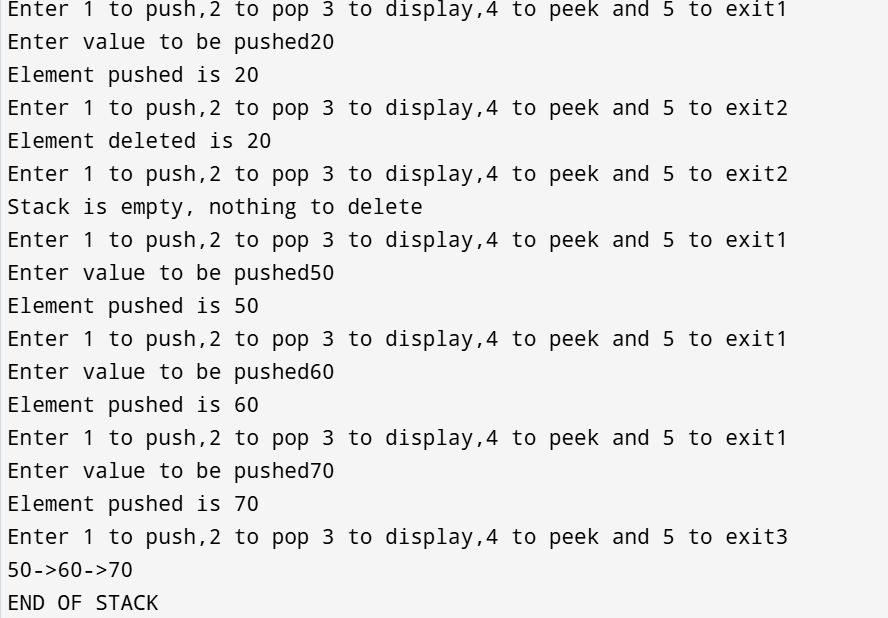
printf("Invalied input");

}

}

}

**OUTPUT:**

****

**6c) WAP to create queue using linked list**

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \*next;

};

struct node \*front = NULL;

struct node \*rear = NULL;

int rearcount = -1;

int frontcount = -1;

struct node\* createnode(int value) {

struct node\* newnode = (struct node\*)malloc(sizeof(struct node));

newnode->data = value;

newnode->next = NULL;

return newnode;

}

void printqueue() {

if (front == NULL) {

printf("Queue is empty\n");

return;

}

struct node\* current = front;

while (current != NULL) {

printf("%d -> ", current->data);

current = current->next;

}

printf("END OF Queue\n");

}

void enqueue(int val) {

if (rearcount == 4) {

printf("Queue is full\n");

} else {

struct node\* newnode = createnode(val);

if (front == NULL) {

front = newnode;

rear = newnode;

rearcount++;

frontcount++;

} else {

rear->next = newnode;

rear = newnode;

rearcount++;

}

}

}

void dequeue() {

if (front == NULL) {

printf("Queue is empty, nothing to delete\n");

} else {

struct node\* temp = front;

front = front->next;

free(temp);

frontcount--;

if (front == NULL) {

rear = NULL;

}

}

}

void main()

{

int n,val;

while(1)

{

printf("Enter 1 to enqueue, 2 to dequeue,3 to display and 4 to exit");

scanf("%d",&n);

if (n==1)

{

printf("Enter value");

scanf("%d",&val);

enqueue(val);

}

else if (n==2)

{

dequeue();

}

else if (n==3)

{

printqueue();

}

else if (n==4)

{

exit(0);

}

else

{

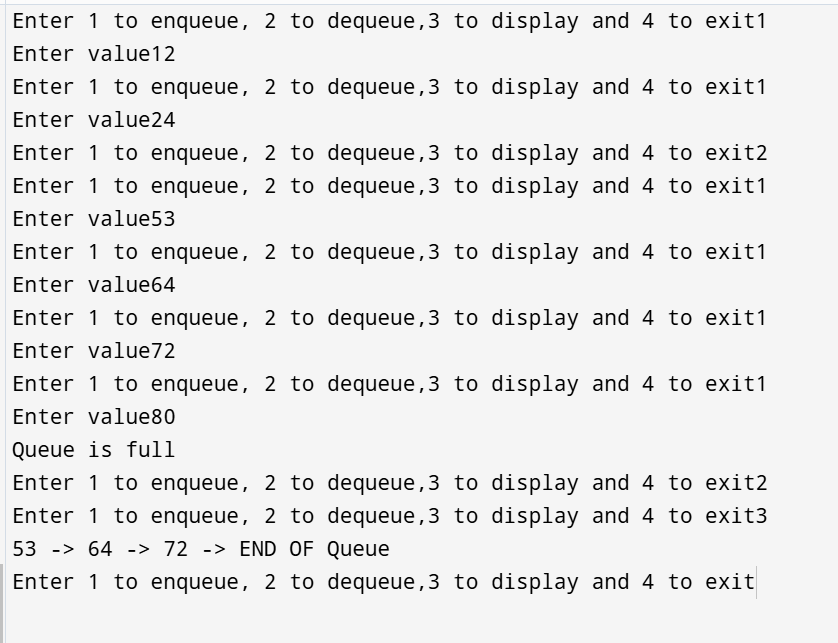
printf("Invalid input");

}

}

}

**OUTPUT:**

****

**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  
Display the contents of the list**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node\* next;

struct node\* prev;

};

struct node\* head=NULL;

struct node\* nextnode=NULL;

struct node\* createnode(int val)

{

struct node\* newnode=(struct node\*)malloc(sizeof(struct node));

newnode->data=val;

newnode->prev=NULL;

newnode->next=NULL;

}

void createdll(int n)

{

int val,i;

for (i=0;i<n;i++)

{

printf("Enter value");

scanf("%d",&val);

struct node\* newnode=createnode(val);

if (head==NULL)

{

head=newnode;

nextnode=newnode;

}

else

{

nextnode->next=newnode;

newnode->prev=nextnode;

nextnode=newnode;

}

}

}

void printdllforward()

{

nextnode=head;

if (head==NULL)

{

printf("list empty\n");

}

while(nextnode->next!=NULL)

{

printf("%d ->",nextnode->data);

nextnode=nextnode->next;

}

printf("%d\n",nextnode->data);

}

void printdllbackward()

{

if (head==NULL)

{

printf("List empty\n");

}

head=nextnode;

while (head->prev!=NULL)

{

printf("%d <-",head->data);

head=head->prev;

}

printf("%d\n",head->data);

}

void insert\_at\_beg(int val)

{

struct node\* newnode=createnode(val);

head->prev=newnode;

newnode->next=head;

head=newnode;

}

void insert\_at\_end(int val)

{

struct node\* newnode=createnode(val);

nextnode->next=newnode;

newnode->prev=nextnode;

nextnode=newnode;

}

void delete\_at\_beg()

{

struct node\* ptr;

ptr=head;

head=head->next;

head->prev=NULL;

free(ptr);

}

void delete\_at\_end()

{

struct node\* ptr;

ptr=nextnode;

nextnode=nextnode->prev;

nextnode->next=NULL;

free(ptr);

}

void main()

{

createdll(3);

printdllforward();

printdllbackward();

insert\_at\_beg(5);

insert\_at\_end(10);

printdllforward();

printdllbackward();

delete\_at\_beg();

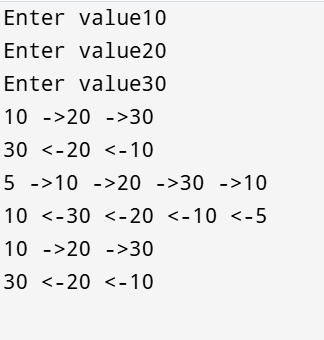
delete\_at\_end();

printdllforward();

printdllbackward();

}

**OUTPUT:**

****

**LAB PROGRAM 8:**

**Binary search tree**

#include<stdio.h>

#include<stdlib.h>

typedef struct Node

{

int data;

struct Node\* left;

struct Node\* right;

}Node;

Node\* createNode(int data)

{

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

newNode->data=data;

newNode->left=NULL;

newNode->right=NULL;

return newNode;

}

Node\* insert(Node\* root,int value)

{

if(root==NULL)

{

return createNode(value);

}

if(value<root->data)

{

root->left=insert(root->left,value);

}

else if(value>root->data)

{

root->right=insert(root->right,value);

}

return root;

}

void postorder(Node\* root)

{

if(root!=NULL)

{

postorder(root->left);

postorder(root->right);

printf("%d \t",root->data);

}

}

void preorder(Node\* root)

{

if(root!=NULL)

{

printf("%d \t",root->data);

preorder(root->left);

preorder(root->right);

}

}

void inorder(Node\* root)

{

if(root!=NULL)

{

inorder(root->left);

printf("%d \t",root->data);

inorder(root->right);

}

}

int main()

{

Node\* root=NULL;

int choice,value;

while(1)

{

printf("\n 1. Inserting a node");

printf("\n 2. postorder");

printf("\n 3. preorder");

printf("\n 4. inorder");

printf("\n 5. Exit");

printf("\n Enter your choice");

scanf("%d",&choice);

switch(choice)

{

case 1:printf("\n enter the data to be added");

scanf("%d",&value);

root=insert(root,value);

break;

case 2:postorder(root);

break;

case 3:preorder(root);

break;

case 4:inorder(root);

break;

case 5:exit(0);

break;

default:printf("\n Invalid Choice");

break;

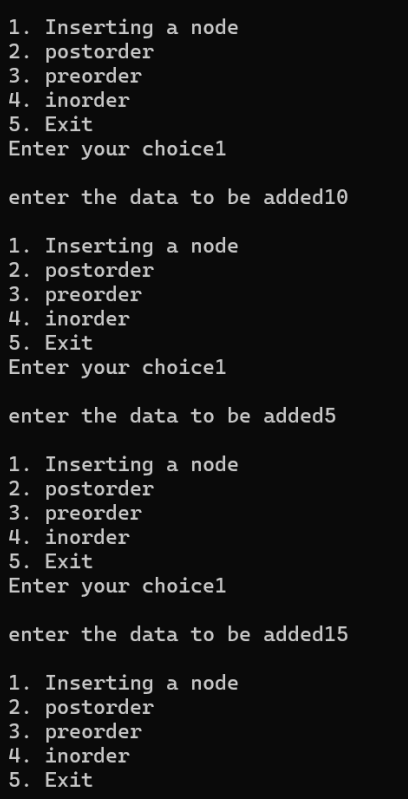
}

}

return 0;

}

**OUTPUT:**

** A screenshot of a computer program

Description automatically generated**

**LEETCODE 3:**

**PALINDROME USING LINKED LIST**

bool isPalindrome(struct ListNode\* head) {

if (head == NULL || head->next == NULL) {

return true;

}

struct ListNode \*a = head, \*b = head;

while (b != NULL && b->next != NULL) {

a = a->next;

b = b->next->next;

}

struct ListNode \*current = head;

int stack[100000];

int index = 0;

while (current != a) {

stack[index++] = current->val;

current = current->next;

}

if (b != NULL) {

a = a->next;

}

while (a != NULL) {

if (a->val != stack[--index]) {

return false;

}

a = a->next;

}

return true;

}

**LAB PROGRAM 9:**

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

#include <stdio.h>

#define MAX 5

void bfs(int adj[][MAX], int visited[], int start) {

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

for (int k = 0; k < MAX; k++)

visited[k] = 0;

queue[++rear] = start;

++front;

visited[start] = 1;

while (rear >= front) {

start = queue[front++];

printf("%d -> ", start);

for (i = 0; i < MAX; i++) {

if (adj[start][i] && visited[i] == 0) {

queue[++rear] = i;

visited[i] = 1;

}

}

}

}

int main() {

int visited[MAX] = {0};

int adj[MAX][MAX], i, j;

printf("Enter the adjacency matrix of the graph (%d x %d):\n", MAX, MAX);

for (i = 0; i < MAX; i++)

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

scanf("%d", &adj[i][j]);

printf("BFS Traversal starting from node 0:\n");

bfs(adj, visited, 0);

return 0;

}

**OUTPUT:**

**A black screen with white text

Description automatically generated**

**LEETCODE 4:**

**SUM OF BINARY TREE**

bool hasPathSum(struct TreeNode\* root, int targetSum) {

if(root==NULL)

return false;

if(root->val==targetSum && root->left==NULL && root->right==NULL)

return true;

else{

return(hasPathSum(root->left,targetSum-root->val) || hasPathSum(root->right,targetSum-root->val));

}

}

**OUTPUT:**