VISVESVARAYA TECHNOLOGICAL UNIVERSITY

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



**LAB REPORT**

**On**

**DATA STRUCTURES (23CS3PCDST)**

**Submitted by:**

**Ridhima Suhane**

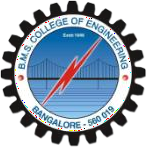
**(1BM23CS266)**

**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)**

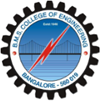
**BENGALURU - 560019**

**September2024-January2025**

**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 **RIDHIMA SUHANE (1BM23CS266)** 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. Lakshmi Neelima 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** | **PageNo.** |
| 1 | LAB PROG-1 | 4-5 |
| 2 | LAB PROG-2 | 6-8 |
| 3 | LAB PROG-3 | 9 |
| 4 | LAB PROG-4 | 10-13 |
| 5 | LAB PROG-5 | 14-17 |
| 6 | LAB PROG-6 | 18-25 |
| 7 | LAB PROG-7 | 26-29 |
| 8 | LAB PROG-8 | 30-32 |
| 9 | LAB PROG-9 | 33-34 |
| 10 | LAB PROG-10 | 35-38 |

**Course outcomes:**

|  |  |
| --- | --- |
| CO1 | Apply the concept of linear and non-linear data structures. |
| CO2 | Analyze data structure operations for a given problem |
| CO3 | Design and Develop solutions using the operations of linear and non linear 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 max 5

int stack[max];

int top = -1;

void push(int value) {

if (top == max - 1) {

printf("Stack overflow!\n");

printf("Cannot push %d into the stack\n", value);

} else {

top++;

stack[top] = value;

printf("%d pushed into the stack\n", value);

}

printf("Element pushed\n");

}

void pop() {

if (top == -1) {

printf("Stack underflow!\n");

printf("Cannot pop from the stack\n");

} else {

int value = stack[top];

printf("%d popped from the stack\n", value);

top--;

}

}

void display() {

if (top == -1) {

printf("Stack is empty\n");

} else {

printf("Stack elements are:\n");

for (int j = top; j >= 0; j--) {

printf("%d\t", stack[j]);

}

printf("\n"); }

}

int main() {

int CHOICE, value;

while (1) {

printf("Choose an operation from below:\n");

printf("1. PUSH\n");

printf("2. POP\n");

printf("3. DISPLAY\n");

printf("4. EXIT\n");

printf("Enter the choice:\n");

scanf("%d", &CHOICE);

switch (CHOICE) {

case 1:

printf("Enter the number:\n");

scanf("%d", &value);

push(value);

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

exit(0);

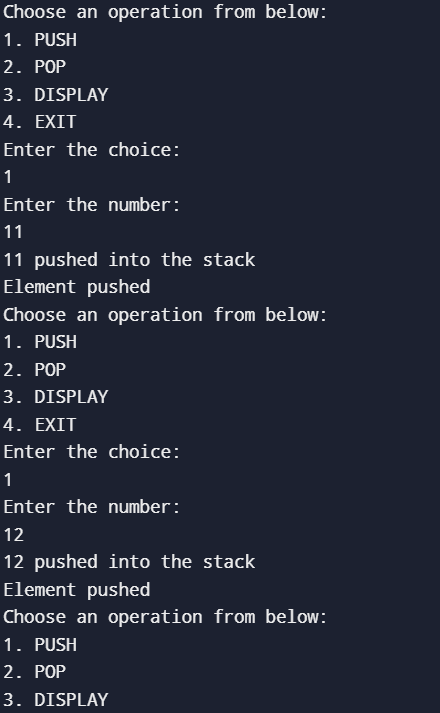
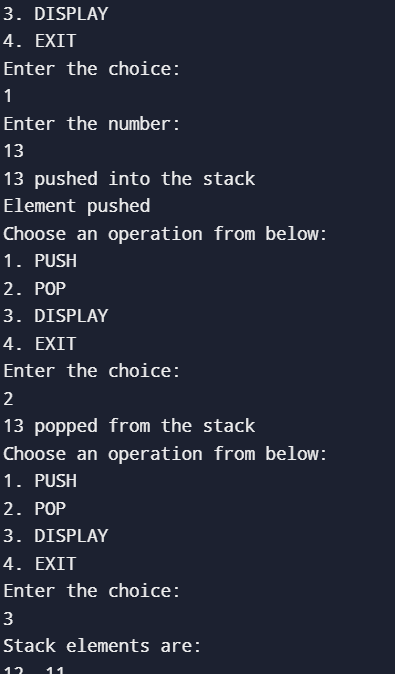
default:

printf("TRY AGAIN!\n");

}

}

**Output**

**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 top=-1,pos=0,length,index0=0;

char symbol,temp,infix[50],postfix[50],stack[50];

void push(char symbol);

char pop();

void infixtopostfix();

int predefined(char);

void main()

{

printf("enter the infix expression:\n");

scanf("%s",infix);

infixtopostfix();

printf("infix expression:%s",infix);

printf("\npostfix expression:%s",postfix);

}

void infixtopostfix()

{

length=strlen(infix);

while(index0<length)

{

symbol=infix[index0];

switch(symbol)

{

case ')':

temp=pop();

while(temp!='(')

{

postfix[pos]=temp;

pos++;

temp=pop();

}

break;

case'(':

push(symbol);

break;

case '\*':

case '/':

case'^':

case '+':

case '-':

while(predefined(stack[top])>=predefined(symbol))

{

temp=pop();

postfix[pos++]=temp;

}

push(symbol);

break;

default:

postfix[pos++]=symbol;

}

index0++;

}

while(top>0)

{

temp=pop();

postfix[pos++]=temp;

}

}

char pop()

{

char symb;

symb=stack[top];

top--;

return symb;

}

void push(char symbol)

{

top++;

stack[top]=symbol;

}

int predefined(char symbol)

{

int p;

switch(symbol)

{

case '^':

p=3;

break;

case '\*':

case '/':

p=2;

break;

case '+':

case '-':

p=1;

break;

case '(':

p=0;

break;

default:

p=-1;

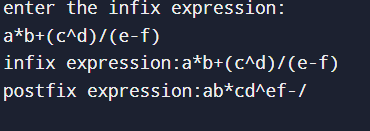
break;

}

return p;

}

**Output**

****

**Lab program 3**

**LEETCODE PROGRAM 1**

**Given an array nums of size n, return *the majority element*.**

**The majority element is the element that appears more than ⌊n / 2⌋ times.**

**You may assume that the majority element always exists in the array.**

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

int i;

int maj=nums[0];

int cnt=0;

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

{

if(cnt==0)

{

maj=nums[i];

}

if(nums[i]==maj)

{

cnt++;

}

else

{

cnt--;

}

}

return maj;

}

**Output**

Input: nums = [3,2,3]

Output: 3

Input: nums = [2,2,1,1,1,2,2]

Output: 2

**Lab program 4**

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

**and queue overflow conditions.**

#include <stdio.h>

#include <stdlib.h>

#define size 3

int Q[size];

int rear=-1;

int front=-1;

void delete1();

void insert1();

void display1();

int main()

{

int choice;

while(1)

{

printf("1.insertion\n");

printf("2.deletion\n");

printf("3.Display\n");

printf("4.Exit\n");

printf("enter your choice\n");

scanf("%d",&choice);

switch(choice)

{

case 1:insert1();

break;

case 2:delete1();

break;

case 3:display1();

break;

case 4:exit(1);

default:

printf("Invalid input\n");

}

}

return 0;

}

void insert1()

{

int item;

if(rear==(size-1))

{

printf("Queue Overflow\n");

}

else

{

if(front ==-1)

front=0;

printf("Enter the element to be inserted\n");

scanf("%d",&item);

rear=rear+1;

Q[rear]=item;

}

}

void delete1()

{

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

{

printf("Queue underflow\n");

return;

}

else

{

printf("Deleted element is:%d\n",Q[front]);

front=front+1;

}

}

void display1()

{

int i;

if(front==-1)

{

printf("Queue is Empty\n");

}

else

{

printf("Queue elements:\n");

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

{

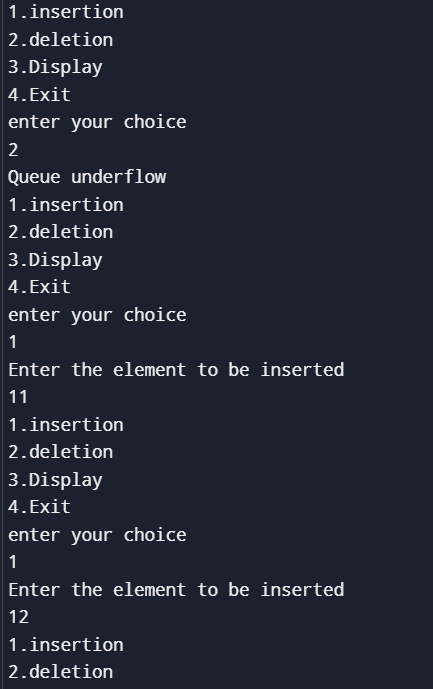
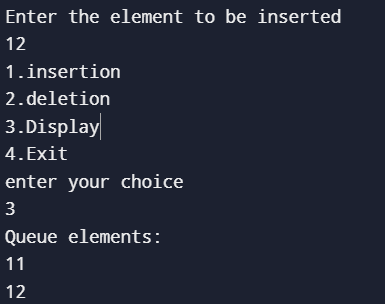
printf("%d\n",Q[i]);

}

}

}

**Output**

** **

**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>

#include <stdlib.h>

#define size 3

int Q[size];

int rear=-1;

int front=-1;

void delete1();

void insert1();

void display1();

int main()

{

int choice;

while(1)

{

printf("1.insertion\n");

printf("2.deletion\n");

printf("3.Display\n");

printf("4.Exit\n");

printf("enter your choice\n");

scanf("%d",&choice);

switch(choice)

{

case 1:insert1();

break;

case 2:delete1();

break;

case 3:display1();

break;

case 4:exit(1);

default:

printf("Invalid input\n");

}

}

return 0;

}

void insert1()

{

int item;

if(rear==(size-1))

{

printf("Queue Overflow\n");

}

else

{

if(front ==-1)

front=0;

printf("Enter the element to be inserted\n");

scanf("%d",&item);

rear=rear+1;

Q[rear]=item;

}

}

void delete1()

{

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

{

printf("Queue underflow\n");

return;

}

else

{

printf("Deleted element is:%d\n",Q[front]);

front=front+1;

}

}

void display1()

{

int i;

if(front==-1)

{

printf("Queue is Empty\n");

}

else{

printf("Queue elements:\n");

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

{

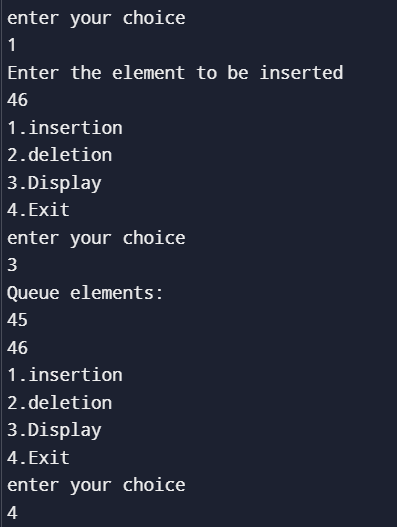
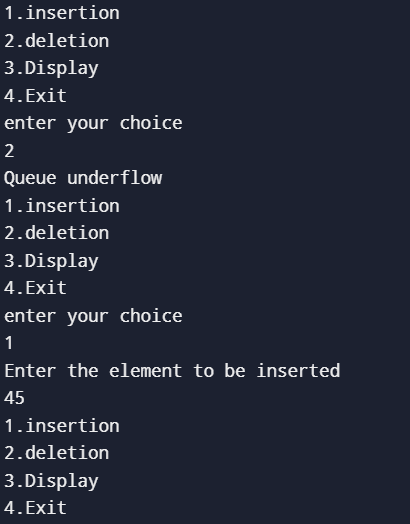
printf("%d\n",Q[i]);

}

}

}

**Output:**

** **

**Lab program 5**

**Write a program to perform insertion and deletion in linked list.**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* createNode(int data) {

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

newNode->data = data;

newNode->next = NULL;

return newNode;

}

void insertAtBeginning(struct Node\*\* head, int data) {

struct Node\* newNode = createNode(data);

newNode->next = \*head;

\*head = newNode;

}

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

struct Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

return;

}

struct Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

}

void insertAfterNode(struct Node\* prevNode, int data) {

if (prevNode == NULL) {

printf("The previous node cannot be NULL\n");

return;

}

struct Node\* newNode = createNode(data);

newNode->next = prevNode->next;

prevNode->next = newNode;

}

void deleteByValue(struct Node\*\* head, int value) {

struct Node\* temp = \*head;

struct Node\* prev = NULL;

if (temp != NULL && temp->data == value) {

\*head = temp->next;

free(temp);

return;

}

while (temp != NULL && temp->data != value) {

prev = temp;

temp = temp->next;

}

if (temp == NULL) {

printf("Node with value %d not found\n", value);

return;

}

prev->next = temp->next;

free(temp);

}

void deleteByPosition(struct Node\*\* head, int position) {

if (\*head == NULL) return;

struct Node\* temp = \*head;

if (position == 0) {

\*head = temp->next;

free(temp);

return;

}

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\* next = temp->next->next;

free(temp->next);

temp->next = next;

}

void printList(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;

while (1) {

printf("\nMenu:\n");

printf("1. Insert at Beginning\n");

printf("2. Insert at End\n");

printf("3. Insert After a Specific Node\n");

printf("4. Delete Node by Value\n");

printf("5. Delete Node by Position\n");

printf("6. Print List\n");

printf("7. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to insert at the beginning: ");

scanf("%d", &data);

insertAtBeginning(&head, data);

break;

case 2:

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

scanf("%d", &data);

insertAtEnd(&head, data);

break;

case 3:

printf("Enter value to insert after: ");

scanf("%d", &data);

struct Node\* temp = head;

while (temp != NULL && temp->data != data) {

temp = temp->next;

}

if (temp != NULL) {

printf("Enter value to insert after node with value %d: ", data);

scanf("%d", &data);

insertAfterNode(temp, data);

} else {

printf("Node with value %d not found.\n", data);

}

break;

case 4:

printf("Enter value to delete: ");

scanf("%d", &data);

deleteByValue(&head, data);

break;

case 5:

printf("Enter position to delete (0-based index): ");

scanf("%d", &position);

deleteByPosition(&head, position);

break;

case 6:

printf("Linked List: ");

printList(head);

break;

case 7:

exit(0);

default:

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

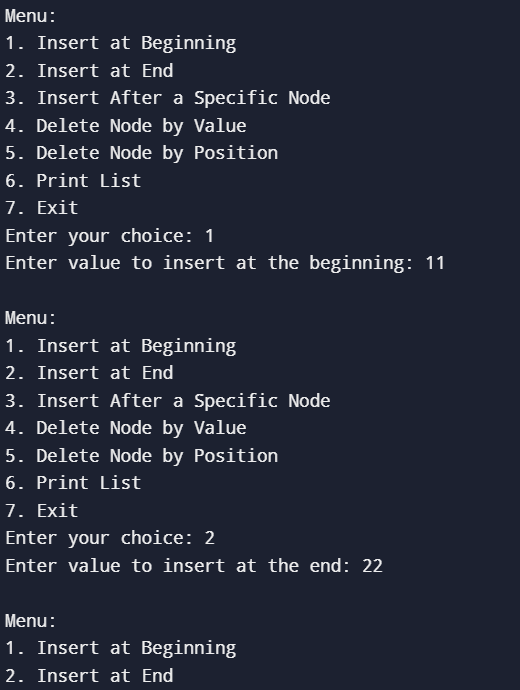
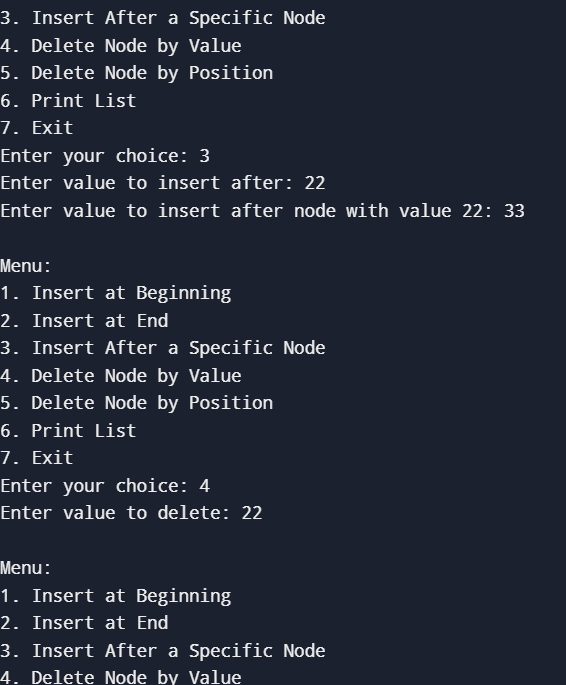
}

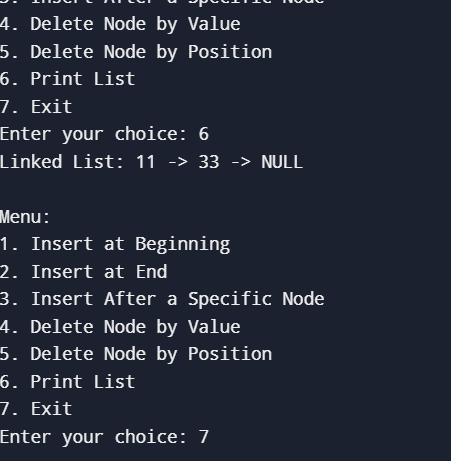
}

return 0;

}

**Output:**

** **

****

**Lab program 6**

**WAP to Implement Single Link List with following operations: Sort the linked list, Reverse**

**the linked list, Concatenation of two linked lists.**

#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 append(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 printList(struct Node\* head) {

struct Node\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

void reverse(struct Node\*\* head) {

struct Node\* prev = NULL;

struct Node\* current = \*head;

struct Node\* next = NULL;

while (current != NULL) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

\*head = prev;

}

void sort(struct Node\* head) {

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

return;

}

struct Node\* current = head;

struct Node\* index = NULL;

int temp;

while (current != NULL) {

index = current->next;

while (index != NULL) {

if (current->data > index->data) {

temp = current->data;

current->data = index->data;

index->data = temp;

}

index = index->next;

}

current = current->next;

}

}

void concatenate(struct Node\*\* head1, struct Node\* head2) {

if (\*head1 == NULL) {

\*head1 = head2;

return;

}

struct Node\* temp = \*head1;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = head2;

}

int main() {

struct Node\* list1 = NULL;

struct Node\* list2 = NULL;

append(&list1, 5);

append(&list1, 3);

append(&list1, 8);

append(&list1, 1);

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

printList(list1);

sort(list1);

printf("Sorted List 1:\n");

printList(list1);

reverse(&list1);

printf("Reversed List 1:\n");

printList(list1);

append(&list2, 7);

append(&list2, 2);

printf("Original List 2:\n");

printList(list2);

concatenate(&list1, list2);

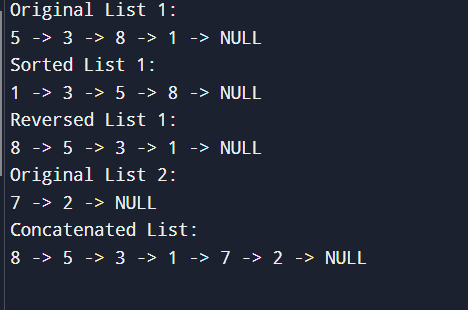
printf("Concatenated List:\n");

printList(list1);

return 0;

}

**Output:**

****

**WAP to Implement Single Link List to simulate Stack and Queue Operations.**

#include <stdio.h>

#include <stdlib.h>

struct node {

int value;

struct node\* next;

};

typedef struct node\* NODE;

NODE getnode() {

NODE ptr = (NODE)malloc(sizeof(struct node));

if (ptr == NULL) {

printf("Memory not allocated\n");

return NULL;

}

return ptr;

}

NODE insert\_beg(int item, NODE first) {

NODE new = getnode();

if (new == NULL) {

return first;

}

new->value = item;

new->next = first;

return new;

}

NODE insert\_last(int item, NODE first) {

NODE new = getnode();

if (new == NULL) {

return first;

}

new->value = item;

new->next = NULL;

if (first == NULL) {

return new;

}

NODE current = first;

while (current->next != NULL) {

current = current->next;

}

current->next = new;

return first;

}

NODE delete\_first(NODE first) {

if (first == NULL) {

printf("Linked list is empty\n");

return NULL;

}

NODE temp = first;

first = first->next;

free(temp);

return first;

}

void display(NODE first) {

if (first == NULL) {

printf("Linked list is empty\n");

return;

}

NODE temp = first;

while (temp != NULL) {

printf("%d -> ", temp->value);

temp = temp->next;

}

printf("NULL\n");

}

int main() {

NODE stackHead = NULL;

NODE queueHead = NULL;

int choice, value;

do {

printf("\nChoose operation:\n");

printf("1. Stack - Push\n");

printf("2. Stack - Pop\n");

printf("3. Queue - Enqueue\n");

printf("4. Queue - Dequeue\n");

printf("5. Display\n");

printf("6. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1: {

printf("Enter value to push onto stack: ");

scanf("%d", &value);

stackHead = insert\_beg(value, stackHead);

break;

}

case 2: {

stackHead = delete\_first(stackHead);

break;

}

case 3: {

printf("Enter value to enqueue into queue: ");

scanf("%d", &value);

queueHead = insert\_last(value, queueHead);

break;

}

case 4: {

queueHead = delete\_first(queueHead);

break;

}

case 5: {

printf("Stack: ");

display(stackHead);

printf("Queue: ");

display(queueHead);

break;

}

case 6: {

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

break;

}

default: {

printf("Invalid choice!\n");

break;

}

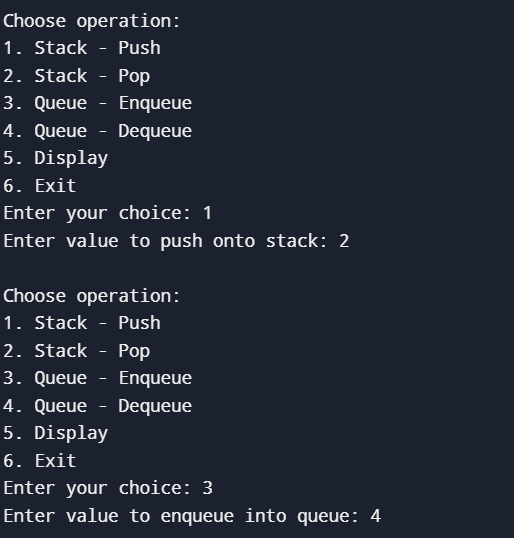
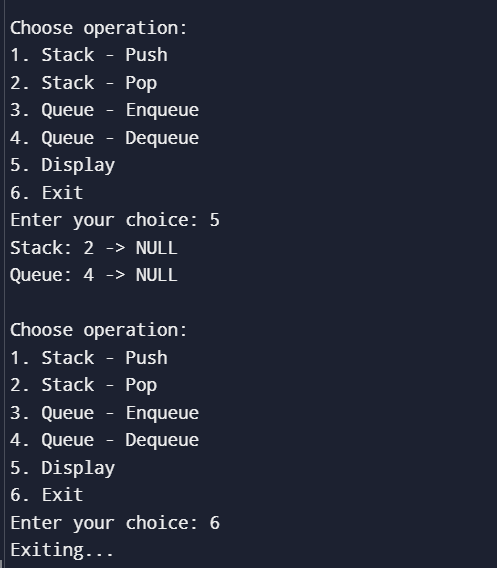
}

} while (choice != 6);

return 0;

}

**Output**

** **

**LEET CODE :**

**Palindrome linked list**

#include <stdbool.h>

struct ListNode\* reverseList(struct ListNode\* head) {

struct ListNode\* prev = NULL;

struct ListNode\* curr = head;

struct ListNode\* next = NULL;

while (curr != NULL) {

next = curr->next;

curr->next = prev;

prev = curr;

curr = next;

}

return prev;

}

bool isPalindrome(struct ListNode\* head) {

if (!head || !head->next) {

return true;

}

struct ListNode\* slow = head;

struct ListNode\* fast = head;

while (fast && fast->next) {

slow = slow->next;

fast = fast->next->next;

}

struct ListNode\* secondHalf = reverseList(slow);

struct ListNode\* firstHalf = head;

struct ListNode\* temp = secondHalf;

bool isPalin = true;

while (secondHalf) {

if (firstHalf->val != secondHalf->val) {

isPalin = false;

break;

}

firstHalf = firstHalf->next;

secondHalf = secondHalf->next;

}

reverseList(temp);

return isPalin;

}

**Output:**

****

**Input:** head = [1,2,2,1]

**Output:** true

**Lab program 7**

**WAP to Implement doubly link list with primitive operations**

1. **Create a doubly linked list.**
2. **Insert a new node to the left of the node.**
3. **Delete the node based on a specific value**
4. **Display the contents of the list**

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

intvalue;

structnode\*next; structnode\*prev;

};

typedefstructnode\*NODE;

NODEgetnode(){ NODE ptr;

ptr=(NODE)malloc(sizeof(struct node));

if(ptr== NULL){

printf("Memorynotallocated\n"); return NULL;

}

returnptr;

}

NODEinsert\_last(intitem,NODEfirst){ NODE new = getnode();

if(new==NULL) returnfirst;

new->value = item; new->next=NULL; new->prev=NULL;

if(first==NULL){ return new;

}

NODEcurrent=first;

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

}

current->next=new;

new->prev=current

return first;

}

NODEinsert\_left(NODEfirst,intitem,intkey){ NODE new, current;

new = getnode(); new->value = item; new->next=NULL; new->prev=NULL;

if (first == NULL) { printf("Listisempty\n"); return first;

}

current= first;

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

}

if(current!=NULL&&current->value==key){ new->next = current;

new->prev=current->prev;

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

}

else{

first=new;//Newnode becomes thefirstif current isthefirstnode

}

current->prev=new; return first;

}else{

printf("Valuenotfound\n"); return first;

}

}

NODEdelete\_value(NODEfirst,intkey){ NODE current = first;

if(first==NULL){

printf("DoublyLinked Listisempty\n");

returnNULL;

}

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

}

if(current!=NULL&&current->value==key){ if (current->prev != NULL) {

current->prev->next=current->next;

}else{

first= current->next;//Update firstif wearedeletingthe firstnode

}

if(current->next!=NULL){

current->next->prev=current->prev;

}

free(current); return first;

}else{

printf("Valuenotfound\n"); return first;

}

}

voiddisplay(NODEfirst){ if (first == NULL) {

printf("Linkedlistisempty\n"); return;

}

NODE temp = first; while(temp!=NULL){

printf("%d<->",temp->value);

temp=temp->next;

}

printf("NULL\n");

}

intmain(){

NODEhead=NULL;

intchoice,item,key; do {

printf("1. Insert Last "); printf("2. Insert at left "); printf("3. Delete Value "); printf("4. Display list "); printf("5. Exit\n"); printf("Enteryourchoice:"); scanf("%d", &choice);

switch(choice){ case 1:

printf("Entervaluetoinsertatend:"); scanf("%d", &item);

head=insert\_last(item,head); break;

case 2:

printf("Entervaluetoinsertatleft:"); scanf("%d", &item);

printf("Enterkeytoinsertbefore:"); scanf("%d", &key);

head=insert\_left(head, item,key);

break; case 3:

printf("Entervaluetodelete:"); scanf("%d", &key);

head=delete\_value(head,key); break;

case 4:

display(head); break;

case 5:

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

default:

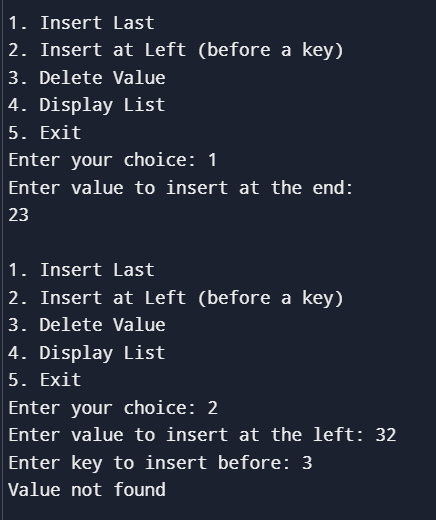
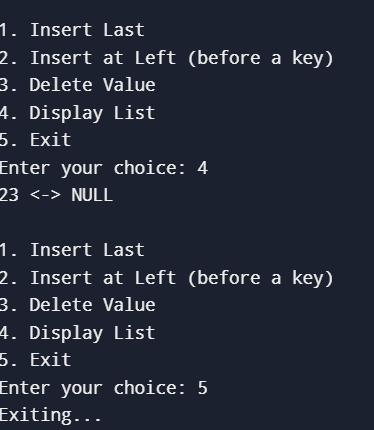
printf("Invalidchoice.Pleasetryagain.\n");

}

}while(choice!=5); return 0;

}

**Output**

** **

**Lab program 8**

**Write a program**

1. **To construct a binary Search tree.**
2. **To traverse the tree using all the method inorder , preorder and post order**
3. **To display the elements in the tree.**

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

Intdata;

struct node \*left; structnode\*right;

};

structnode\*newNode(intdata) {

structnode\*node=(structnode\*)malloc(sizeof(structnode)); node->data = data;

node->left=node->right=NULL; return node;

}

structnode\*insert(structnode\*root,intdata){ if (root == NULL)

returnnewNode(data); if (data < root->data)

root->left=insert(root->left,data); else if (data > root->data)

root->right=insert(root->right,data); return root;

}

voidinorder(structnode\*root){ if (root != NULL) {

inorder(root->left); printf("%d",root->data); inorder(root->right);

}

}

voidpreorder(structnode\*root){ if (root != NULL) {

printf("%d",root->data); preorder(root->left); preorder(root->right);

}

}

voidpostorder(structnode\*root){ if (root != NULL) {

postorder(root->left); postorder(root->right); printf("%d",root->data);

}

}

voiddisplay(structnode\*root,intchoice){ switch (choice) {

case 1:

printf("\nIn-ordertraversal:"); inorder(root);

break; case 2:

printf("\nPre-ordertraversal:");

preorder(root);

break;

case 3:

printf("\nPost-order traversal: "); postorder(root);

break; default:

printf("Invalidchoice\n"); break;

}

}

intmain(){

structnode\*root=NULL; int n, data, choice;

printf("EnterthenumberofnodestoinsertintheBST:"); scanf("%d", &n);

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

printf("Entervaluefornode%d:",i+1); scanf("%d", &data);

root=insert(root,data);

}

while(1){

printf("\nChoosethetypeoftraversal:\n"); printf("1.In-order\t");

printf("2.Pre-order\t"); printf("3.Post-order\t"); printf("4.Exit\n"); printf("Enteryourchoice:"); scanf("%d", &choice);

if (choice == 4) { printf("Exiting...\n"); break;

}

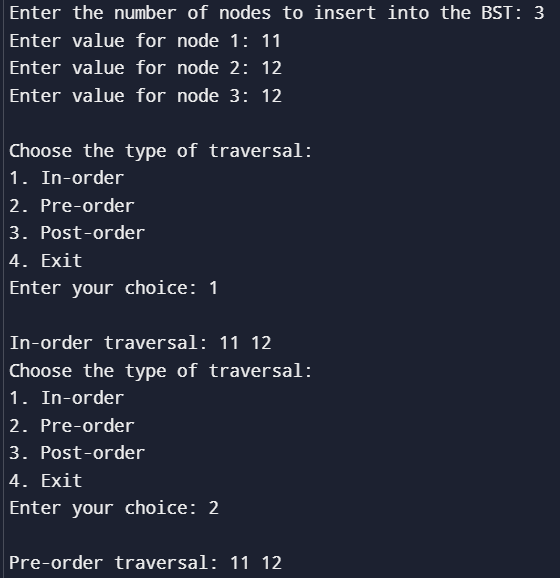
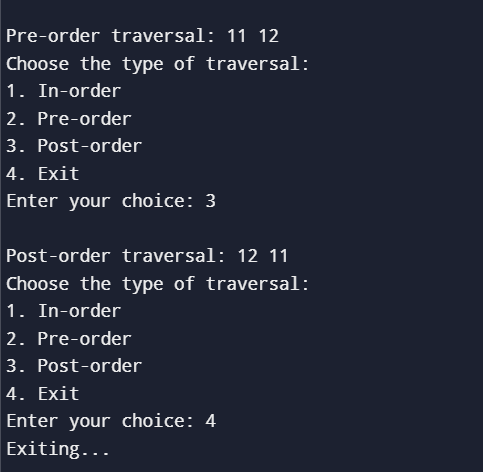
display(root,choice);

}

return 0;

}

**Output**

** **

**Lab program 9**

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

#include<stdio.h>

Int a[10][10],vis[10],parent[10],n; char nodes[10];

void bfs(int v){

intq[10],f=0,r=0,u,i; q[r]=v;

vis[v] =1;

parent[v] = -1; printf("%c",nodes[v]); while (f <= r) {

u=q[f]; f++;

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

if(a[u][i]==1&&vis[i]==0){ vis[i] = 1;

r++;

q[r] = i; parent[i]=u;

printf("%c",nodes[i]);

}

elseif(a[u][i]==1&&vis[i]==1&&parent[u]!=i){ printf("\nCycle detected!\n");

return;

}

}

}

printf("\n");

}

intisConnected(){

for(inti=0;i<n;i++){ if (vis[i] == 0) {

return 0;

}

}

return 1;

}

intmain(){ int src;

printf("Enternumberofvertices:"); scanf("%d", &n);

printf("Enternodelabels(characters)foreachvertex:\n"); for (int i = 0; i < n; i++) {

printf("Node%d: ",i+1);

scanf("%c",&nodes[i]);

}

printf("Enteradjacencymatrix(0or1):\n"); for (int i = 0; i < n; i++) {

for(intj=0;j<n;j++){ scanf("%d", &a[i][j]);

}

vis[i]=0;

parent[i]=-1;

}

printf("Entersourcevertex (bylabel):");

charsrc\_label;

scanf("%c",&src\_label); int src\_index = -1;

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

if(nodes[i]==src\_label){ src\_index = i;

break;

}

}

if (src\_index == -1) { printf("Invalidsourcevertex.\n"); return 0;

}

printf("BFSTraversalstartingfromvertex'%c':\n",src\_label); bfs(src\_index);

if(isConnected()){

printf("Thegraphisconnected.\n");

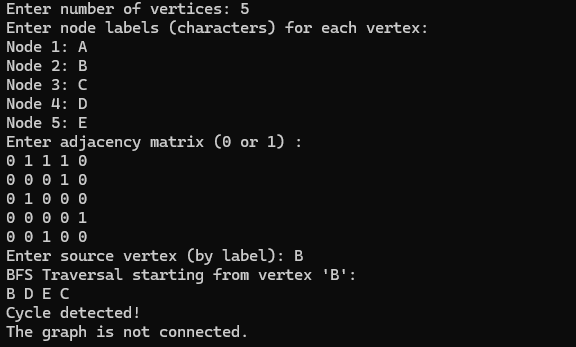
}else{

printf("Thegraph isnot connected.\n");

}

return 0;

}

**Output**

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

#include <stdio.h>

int a[10][10], vis[10], parent[10], n; char nodes[10];

int dfs(int v)

{

vis[v] = 1;

printf("%c ", nodes[v]); for (int j = 0; j < n; j++)

{

if (a[v][j] == 1 && vis[j] == 0)

{

parent[j] = v; if (dfs(j))

return 1;

}

else if (a[v][j] == 1 && vis[j] == 1 && parent[v] != j)

{

printf("\nCycle detected!\n"); return 1;

}

}

return 0;

}

int isConnected()

{

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

{

if (vis[i] == 0)

return 0;

}

return 1;

}

int main()

{

printf("Enter number of vertices: "); scanf("%d", &n);

printf("Enter the characters for the nodes (e.g., A, B, C, etc.):\n"); for (int i = 0; i < n; i++)

{

printf("Node %d: ", i + 1);

scanf(" %c", &nodes[i]);

}

printf("Enter adjacency matrix (0 or 1) :\n"); for (int i = 0; i < n; i++)

{

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

{

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

}

vis[i] = 0;

parent[i] = -1;

}

printf("DFS Traversal: "); for (int i = 0; i < n; i++)

{

if (vis[i] == 0)

{

if (dfs(i))

{

break;

}

}

}

if (isConnected())

{

printf("\nThe graph is connected.\n");

}

else

{

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

}

return 0;

}

**Output**

****

**Lab program 10**

**Hashing using linear probing**

#include <stdio.h>

#include<stdlib.h>

#define TABLE\_SIZE 10

int h[TABLE\_SIZE]={NULL};

void insert()

{

int key,index,i,flag=0,hkey;

printf("\nenter a value to insert into hash table\n");

scanf("%d",&key);

hkey=key%TABLE\_SIZE;

for(i=0;i<TABLE\_SIZE;i++)

{

index=(hkey+i)%TABLE\_SIZE;

if(h[index] == NULL)

{

h[index]=key;

break;

}

}

printf("No of probes for %d is %d", key,i+1);

if(i == TABLE\_SIZE)

printf("\nelement cannot be inserted\n");

}

void search()

{

int key,index,i,flag=0,hkey;

printf("\nenter search element\n");

scanf("%d",&key);

hkey=key%TABLE\_SIZE;

for(i=0;i<TABLE\_SIZE; i++)

{

index=(hkey+i)%TABLE\_SIZE;

if(h[index]==key)

{

printf("value is found at index %d",index);

break;

}

}

if(i == TABLE\_SIZE)

printf("\n value is not found\n");

}

void display()

{

int i;

printf("\nelements in the hash table are \n");

for(i=0;i< TABLE\_SIZE; i++)

printf("\nat index %d \t value = %d",i,h[i]);

}

main()

{ int opt,i;

while(1)

{ printf("\nPress 1. Insert\t 2. Display \t3. Search \t4.Exit \n");

scanf("%d",&opt);

switch(opt)

{

case 1:insert();

break;

case 2:display();

break;

case 3:search();

break;

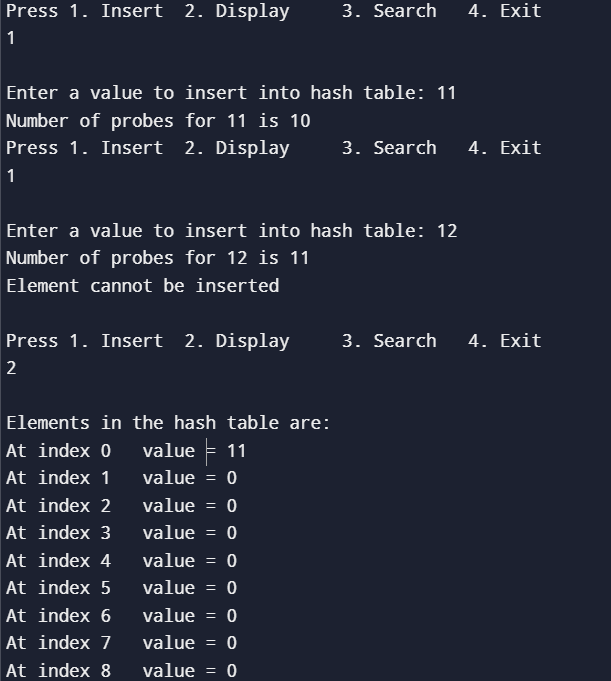
case 4:exit(0);

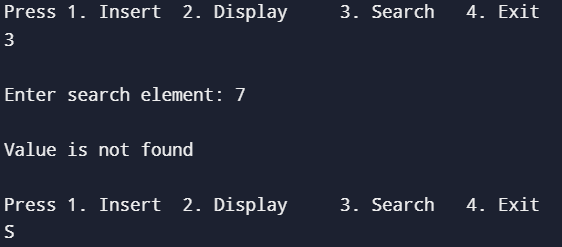
}

}

}

**output**

****

****