**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

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

****

**LAB REPORT**

**On**

**DATA STRUCTURES (23CS3PCDST)**

**Submitted by**

**MAHAMAD AZIZ ANSARI (1BM23CS174)**

**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 MAHAMAD AZIZ ANSARI **(1BM23CS174)**, who is a bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2024-25. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - **(23CS3PCDST)** work prescribed for the said degree.

**Prof. Namratha M** **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 | Program to simulate the working of stack | 4 |
| 2 | Program to convert a given valid parenthesized infix arithmetic expression to postfix expression | 6 |
| 3 | Program to simulate the working of a queue  LeetCode: Queue using Stack | 9 |
| 4 | Program to simulate the working of a circular queue  LeetCode: Valid Parentheses | 14 |
| 5 | Program to Implement Singly Linked List (Create, Insert, Delete, Display) | 19 |
| 6 | Linked list operations (Sort, Reverse, Concatenate)  Implementation of Queue and Stack using Linked List | 27 |
| 7 | Program to Implement doubly link list  LeetCode: Finding middle element of a linked list | 38 |
| 8 | Program to Implement Binary Search Tree | 45 |
| 9a | BFS traversal | 49 |
| 9b | DFS traversal and other functions | 51 |

**Course outcomes:**

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

**Lab program 1:**

**Write a program to simulate the working of stack using an array with the following:**

**a) Push**

**b) Pop**

**c) Display**

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

#include <stdio.h>

#define SIZE 5

void push(int arr[SIZE], int \*top, int ele) {

if (\*top == SIZE - 1) {

printf("Overflow\n");

} else {

(\*top)++;

arr[\*top] = ele;

}

}

int pop(int arr[SIZE], int \*top) {

if (\*top == -1) {

printf("Underflow\n");

return 0;

} else {

int res = arr[\*top];

(\*top)--;

return res;

}

}

void display(int arr[SIZE], int top) {

if (top == -1) {

printf("Empty stack\n");

} else {

printf("Stack elements: ");

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

printf("%d ", arr[i]);

}

printf("\n");

}

}

int main() {

int a[SIZE], ch, ele, top = -1, z = 1;

while (z) {

printf("Enter the choice:\n1. push\n2. pop\n3. display\n");

scanf("%d", &ch);

switch (ch) {

case 1:

printf("Enter element to push: ");

scanf("%d", &ele);

push(a, &top, ele);

break;

case 2:

printf("Last item popped: %d\n", pop(a, &top));

break;

case 3:

display(a, top);

break;

default:

printf("Error occurred\n");

break;

}

printf("Want to perform more operations? (1/0): ");

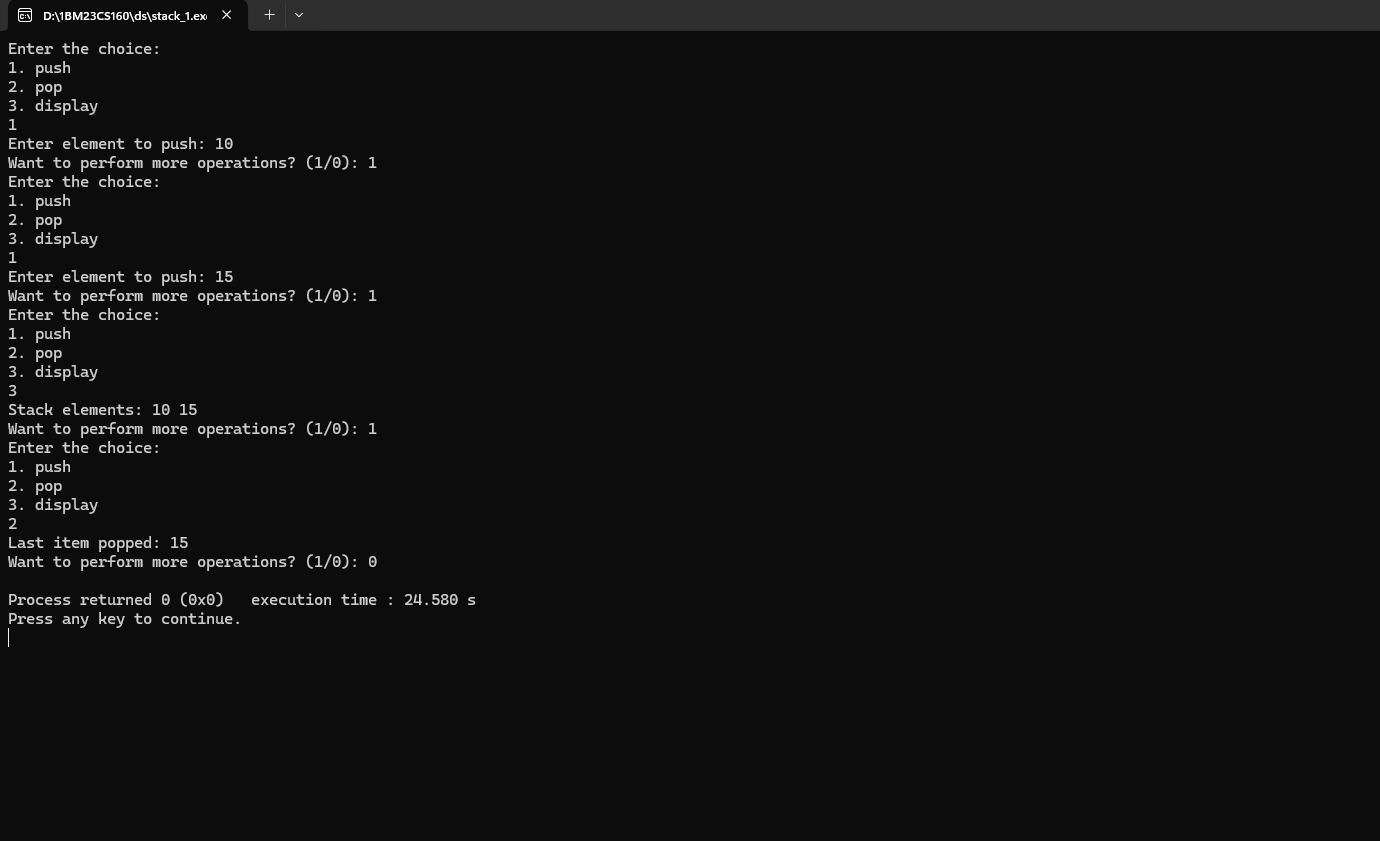
scanf("%d", &z);

}

return 0;

}

**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<ctype.h>;

char stack[100];

int top=-1;

void push(char ele)

{

top++;

stack[top]=ele;

}

char pop()

{

return (stack[top--]);

}

int pr(char op)

{

switch(op)

{

case '#' : return 0;

case '(' : return 1;

case '+' :

case '-':

return 2;

case '\*' :

case '/' :

return 3;

default:

return 0;

}

}

void main()

{

char infix[100], postfix[100];

int i=0;

char ch;

printf("Enter your infix expression:");

scanf("%s",infix);

push('#');

while (infix[i]!='\0'){

if (isalpha(infix[i]))

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

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

push(infix[i]);

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

{

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

{

ch=pop();

printf("%c",ch);

}

pop();

}

else

{

while ((stack[top]!='#') && (pr(infix[i])<=pr(stack[top])))

{

ch=pop();

printf("%c",ch);

}

push(infix[i]);

}

i++;

}

for(i=top;i!=0;i--)

{

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

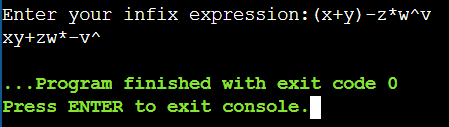
printf ("\n There was an issue with the expression...");

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

}

}

**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>

#define SIZE 5

void insert(int \*front, int \*rear, int q[], int e){

if(\*rear==SIZE-1){

printf("Overflow");

}

else{

q[++(\*rear)]=e;

}

}

int del(int \*front, int \*rear, int q[]){

if(\*front>\*rear){

printf("Underflow");

return -1;

}

else{

return ((q[(\*front)++]));

}

}

void display(int \*front, int \*rear, int q[]){

if(\*rear == -1){

printf("Empty");

}

else{

printf("Items: ");

for(int i = \*front; i<=\*rear; i++){

printf("%d ",q[i]);

}

}

}

void main(){

int q[SIZE];

int front = 0;

int rear = -1;

int ch;

int e;

while(1){

printf("\nMenu:\n1.Insert\n2.Delete\n3.Display\n");

scanf("%d",&ch);

switch(ch){

case 1:

printf("Enter the element: ");

scanf("%d",&e);

insert(&front,&rear,q,e);

break;

case 2:

if(del(&front,&rear,q)==-1)

{

}

else

printf("%d is deleted.\n",del(&front,&rear,q));

break;

case 3:

display(&front, &rear, q);

break;

case 4:

exit(0);

default:

printf("Invalid choice");

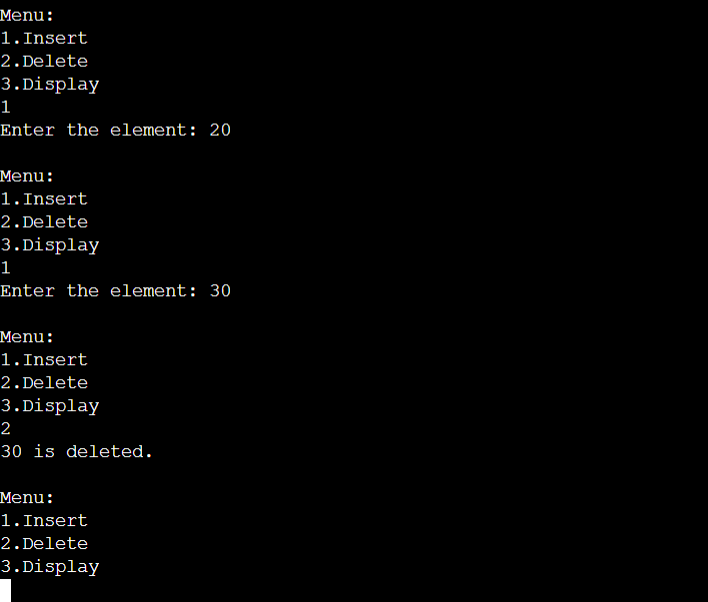
break;

}

}

}

**Output:**

****

**LeetCode 1:**

**Implement Queue using Stacks**

class MyQueue {

public:

stack<int> stk1;

stack<int> stk2;

MyQueue() {

}

void push(int x) {

stk1.push(x);

}

int pop() {

while(stk1.size()>1)

{

stk2.push(stk1.top());

stk1.pop();

}

int ans = stk1.top();

stk1.pop();

while(!stk2.empty())

{

stk1.push(stk2.top());

stk2.pop();

}

return ans;

}

int peek() {

while(stk1.size()>1)

{

stk2.push(stk1.top());

stk1.pop();

}

int ans = stk1.top();

while(!stk2.empty())

{

stk1.push(stk2.top());

stk2.pop();

}

return ans;

}

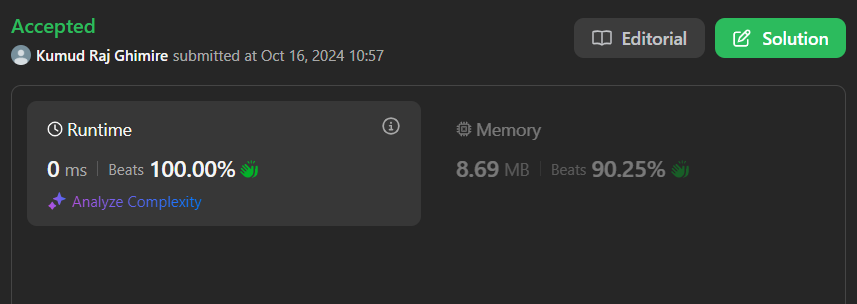
bool empty() {

return stk1.empty();

}

};

**Output:**

****

**Lab Program 4:**

**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>

#include <stdlib.h>

#define SIZE 5

void insert(int \*front, int \*rear, int q[], int e) {

if (\*front == -1 && \*rear == -1) {

\*front = \*rear = 0;

q[\*rear] = e;

}

else if ((\*rear + 1) % SIZE == \*front) {

printf("Overflow\n");

}

else {

\*rear = (\*rear + 1) % SIZE;

q[\*rear] = e;

}

}

void del(int \*front, int \*rear, int q[]) {

if (\*front == -1 && \*rear == -1) {

printf("Underflow\n");

}

else if (\*front == \*rear) {

printf("%d is popped\n", q[\*front]);

\*front = \*rear = -1;

}

else {

printf("%d is popped\n", q[\*front]);

\*front = (\*front + 1) % SIZE;

}

}

void display(int \*front, int \*rear, int q[]) {

if (\*front == -1 && \*rear == -1) {

printf("Queue is empty\n");

} else {

int i = \*front;

printf("Items: ");

while(i!= \*rear){

printf("%d ",q[i]);

i = (i + 1) % SIZE;

}

}

printf("%d\n",q[\*rear]);

}

int main() {

int front = -1, rear = -1;

int q[SIZE];

int ch;

int e;

while (1) {

printf("Enter the choice:\n1. Insert\n2. Delete\n3. Display\n4. Exit\n");

scanf("%d", &ch);

switch (ch) {

case 1:

printf("Enter element to insert: ");

scanf("%d", &e);

insert(&front, &rear, q, e);

break;

case 2:

del(&front, &rear, q);

break;

case 3:

display(&front, &rear, q);

break;

case 4:

exit(0);

default:

printf("Invalid choice\n");

break;

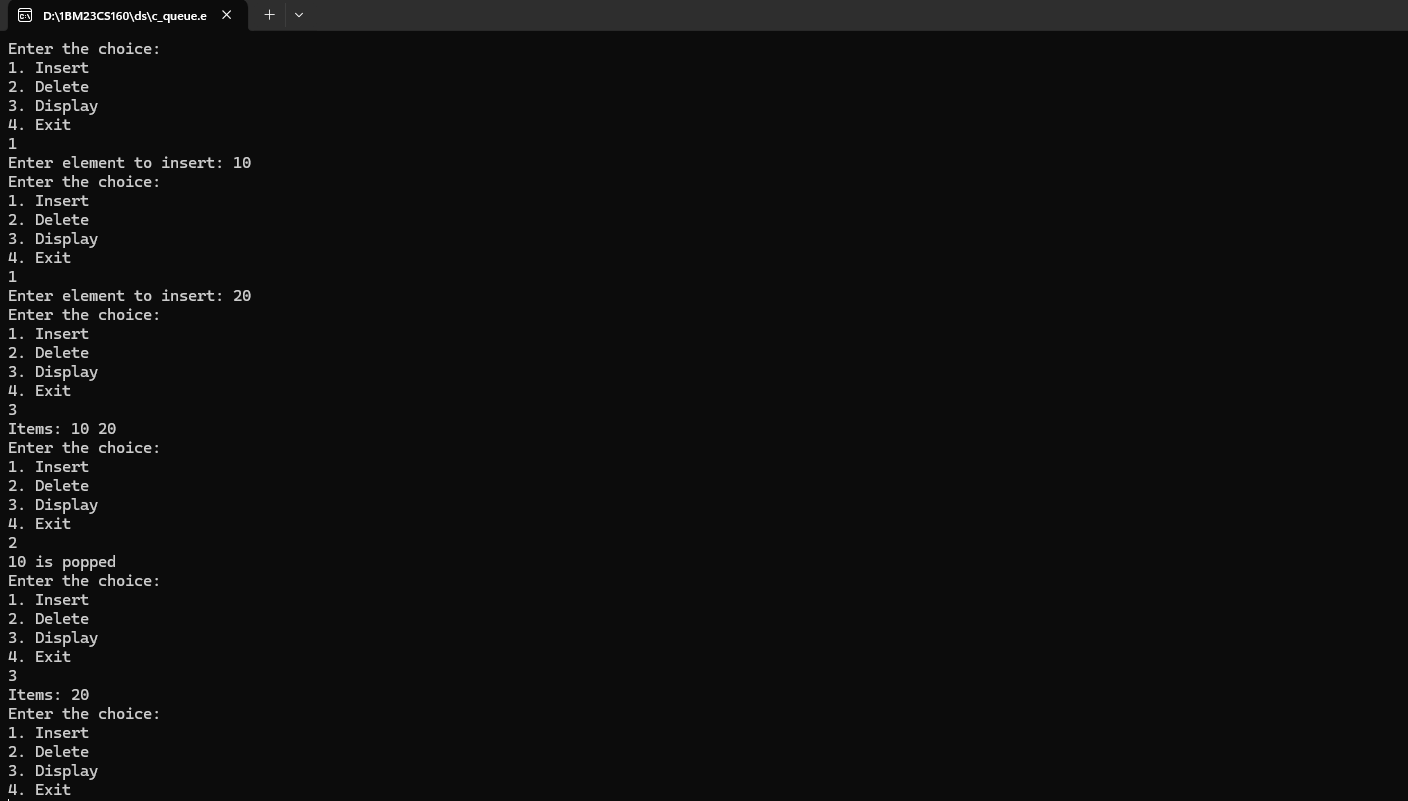
}

}

return 0;

}

**Output:**



**LeetCode 2:**

**Valid Parentheses**

class Solution:

def isValid(self, s: str) -> bool:

lst = []

valid = {')': '(', '}': '{', ']': '['}

for i in s:

if i == '(' or i == '{' or i == '[':

lst.append(i)

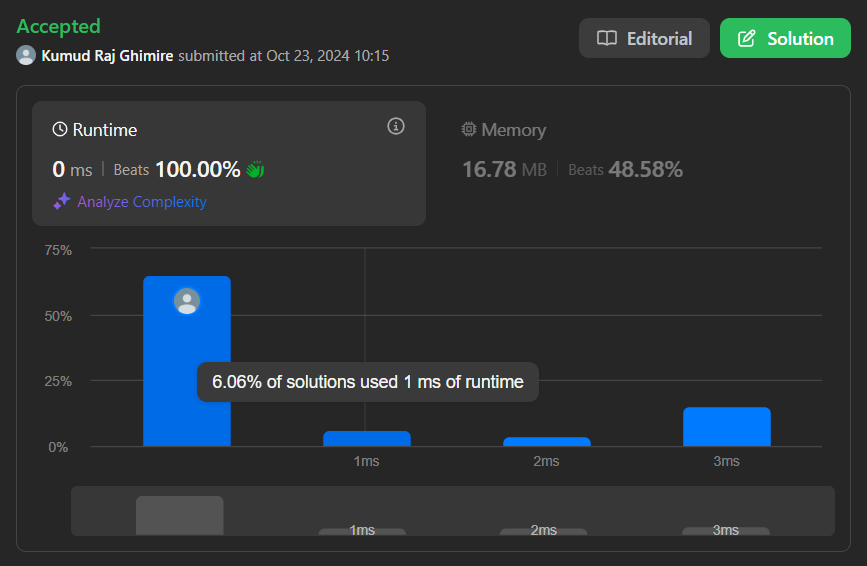
elif i == ')' or i == '}' or i == ']':

if not lst or lst.pop() != valid[i]:

return False

return len(lst) == 0

**Output:**



**Lab Program 5:**

**Program to Implement Singly Linked List (Create, Insert, Delete, Display)**

#include<stdio.h>

#include<stdlib.h>

//#include<alloc.h>

struct Node{

int data;

struct Node\* next;

};

struct Node\* createNode(int data){

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

newNode->data=data;

newNode->next=NULL;

return newNode;

};

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

struct Node\* newNode = createNode(data);

newNode->next = \*head;

\*head = newNode;

}

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

struct Node\* newNode = createNode(data);

if(\*head==NULL){

\*head = newNode;

return;

}

struct Node\* last = \*head;

while(last->next!=NULL){

last=last->next;

}

last->next=newNode;

}

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

if(position==0){

insertAtFirst(head,data);

return;

}

struct Node\* newNode = createNode(data);

struct Node\* current = \*head;

for(int i=0;i<position-1&&current!=NULL;i++){

current=current->next;

}

if(current==NULL){

printf("position exceeds length of list\n");

free(newNode);

return;

}

newNode->next = current -> next;

current -> next = newNode;

}

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

if(\*head==NULL){

printf("List empty\n");

return;

}

struct Node\* temp =\*head;

\*head= (\*head)->next;

free(temp);

}

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

if(\*head==NULL){

printf("List Empty");

return;

}

if((\*head)->next==NULL){

free(\*head);

return;

}

struct Node\* second\_last=\*head;

while(second\_last->next->next!=NULL){

second\_last=second\_last->next;

}

free(second\_last->next);

second\_last->next=NULL;

}

void deleteElement(struct Node\*\* head, int key){

struct Node\* current = \*head;

struct Node\* previous = NULL;

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

\*head = current -> next;

free(current);

return;

}

while(current!=NULL && current->data!=key){

previous = current;

current = current -> next;

}

if(current==NULL){

printf("Element not found");

return;

}

previous->next = current ->next;

free(current);

}

void display(struct Node\* head){

struct Node\* current = head;

while(current!=NULL){

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

current=current->next;

}

printf("NULL");

}

void main(){

struct Node\* head = NULL;

int ch,e,i;

while(1){

printf("\n\nMenu\n1.Insert at first\n2.Insert at end\n3.Insert at index\n4.Delete first\n5.Delete last\n6.Delete anywhere\n7.Display\n8.Exit\n");

scanf("%d",&ch);

switch(ch){

case 1:

printf("Enter element to insert: ");

scanf("%d",&e);

insertAtFirst(&head,e);

break;

case 2:

printf("Enter element to insert: ");

scanf("%d",&e);

insertAtEnd(&head,e);

break;

case 3:

printf("Enter element to insert: ");

scanf("%d",&e);

printf("Enter index to insert at: ");

scanf("%d",&i);

insertAtPosition(&head,e,i);

break;

case 4:

printf("Element at first deleted!\n");

deleteFirst(&head);

break;

case 5:

printf("Element at last deleted!\n");

deleteLast(&head);

break;

case 6:

printf("Enter element to delete: ");

scanf("%d",&e);

deleteElement(&head,e);

break;

case 7:

display(head);

break;

case 8:

exit(0);

default:

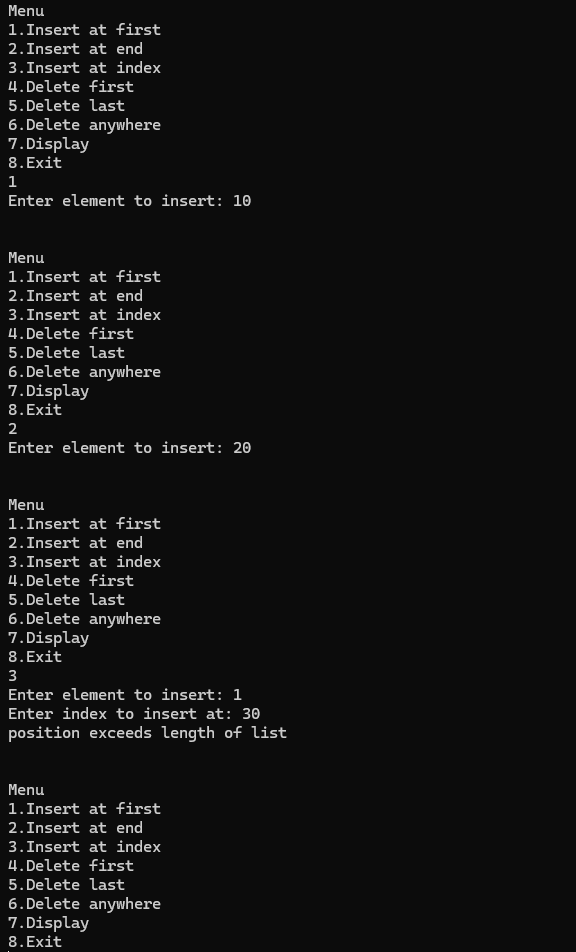
printf("Invalid choice");

}

}

}

**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>

//#include<alloc.h>

struct Node{

int data;

struct Node\* next;

};

typedef struct Node\* NODE;

struct Node\* createNode(int data){

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

newNode->data=data;

newNode->next=NULL;

return newNode;

};

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

struct Node\* newNode = createNode(data);

newNode->next = \*head;

\*head = newNode;

}

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

struct Node\* newNode = createNode(data);

if(\*head==NULL){

\*head = newNode;

return;

}

struct Node\* last = \*head;

while(last->next!=NULL){

last=last->next;

}

last->next=newNode;

}

void display(struct Node\* head){

struct Node\* current = head;

while(current!=NULL){

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

current=current->next;

}

printf("NULL");

}

void sort(NODE first){

int x;

NODE temp1, temp2;

//temp1=first;

//temp2=temp1->next;

for (temp1 = first; temp1 != NULL && temp1->next != NULL; temp1 = temp1->next){

for (temp2 = temp1->next; temp2 != NULL; temp2 = temp2->next){

if((temp1->data)>(temp2->data)){

x=temp1->data;

temp1->data=temp2->data;

temp2->data=x;

}

//temp2=temp2->next;

}

//temp1=temp1->next;

}

}

NODE Reverse(NODE first){

NODE current,temp;

current = NULL;

while(first!=NULL){

temp=first;

first=first->next;

temp->next=current;

current=temp;

}

return current;

}

NODE Concatenate(NODE first1, NODE first2){

NODE last1;

if(first1==NULL && first2==NULL){

return NULL;

}

if(first1==NULL){

return first2;

}

if(first2==NULL){

return first1;

}

last1=first1;

while(last1->next!=NULL){

last1=last1->next;

}

last1->next=first2;

return first1;

}

void main(){

NODE head = NULL;

NODE newhead = NULL;

int ch;

insertEnd(&head,10);

insertEnd(&head,9);

insertEnd(&head,100);

insertEnd(&head,25);

insertEnd(&head,13);

insertEnd(&newhead,133);

insertEnd(&newhead,134);

insertEnd(&newhead,135);

while(1){

printf("\nEnter your choice: \n1.original\n2.sort\n3.reverse\n4.concatenate\n5.exit\n");

scanf("%d",&ch);

switch(ch){

case 1:

printf("Original:\n");

display(head);

break;

case 2:

printf("\nSorted:\n");

sort(head);

display(head);

break;

case 3:

printf("\nReverse:\n");

display(Reverse(head));

break;

case 4:

printf("\nConcatenated:\n");

display(Concatenate(head,newhead));

break;

case 5:

exit(0);

default:

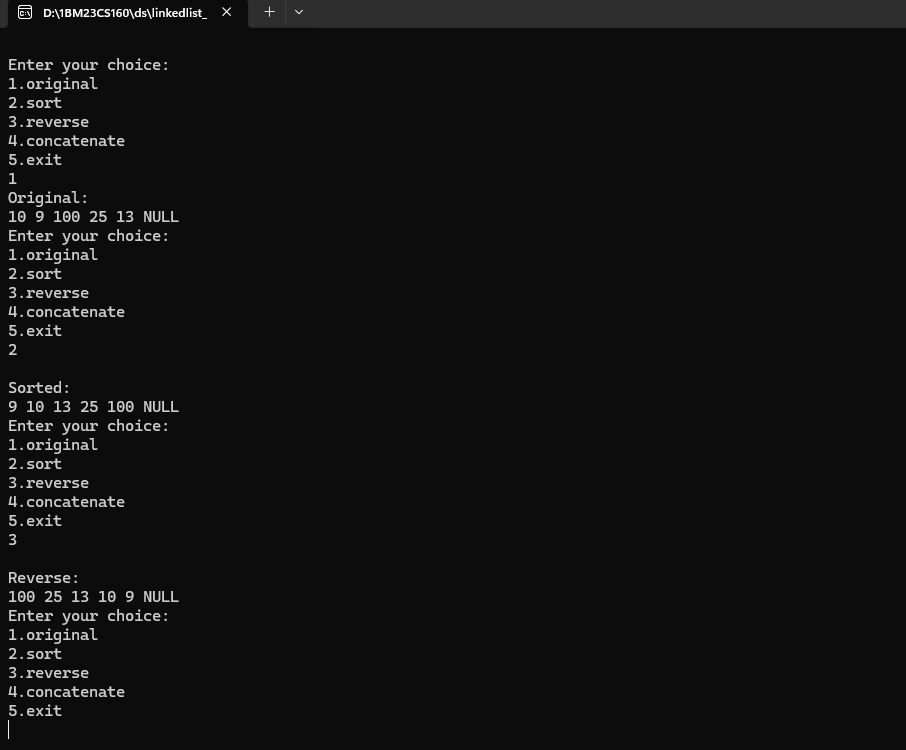
printf("Invalid choice");

}

}

}

**Output:**

****

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

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

typedef struct Node\* NODE;

struct Node\* createNode(int data) {

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

newNode->data = data;

newNode->next = NULL;

return newNode;

}

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

struct Node\* newNode = createNode(data);

newNode->next = \*head;

\*head = newNode;

}

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

if (\*head == NULL) {

printf("List empty\n");

return;

}

struct Node\* temp = \*head;

\*head = (\*head)->next;

free(temp);

}

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

if (\*head == NULL) {

printf("List Empty\n");

return;

}

if ((\*head)->next == NULL) {

free(\*head);

\*head = NULL;

return;

}

struct Node\* second\_last = \*head;

while (second\_last->next->next != NULL) {

second\_last = second\_last->next;

}

free(second\_last->next);

second\_last->next = NULL;

}

void display(struct Node\* head) {

struct Node\* current = head;

while (current != NULL) {

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

current = current->next;

}

printf("NULL\n");

}

void push(NODE\* head, int e) {

insertAtFirst(head, e);

}

void pop(NODE\* head) {

deleteFirst(head);

}

void enqueue(NODE\* head, int e) {

insertAtFirst(head, e);

}

void dequeue(NODE\* head) {

deleteLast(head);

}

void main() {

NODE stack = NULL;

NODE queue = NULL;

int choice, value;

while (1) {

printf("\n1. Push\n2. Pop\n3. Display Stack\n4. Enqueue\n5. Dequeue\n6. Display Queue\n7. Exit\nEnter choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to push: ");

scanf("%d", &value);

push(&stack, value);

break;

case 2:

pop(&stack);

break;

case 3:

printf("Stack: ");

display(stack);

break;

case 4:

printf("Enter value to enqueue: ");

scanf("%d", &value);

enqueue(&queue, value);

break;

case 5:

dequeue(&queue);

break;

case 6:

printf("Queue: ");

display(queue);

break;

case 7:

exit(0);

default:

printf("Invalid choice\n");

}

}

}

**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 d) Display the contents of the list**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int value;

struct Node\* next;

struct Node\* prev;

};

typedef struct Node\* NODE;

NODE getNode(int data) {

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

newNode->value = data;

newNode->next = NULL;

newNode->prev = NULL;

return newNode;

}

NODE insert\_left\_value(NODE first, int item, int key) {

NODE new = getNode(item), current = first;

if (first == NULL) {

return new;

}

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;

}

current->prev = new;

} else {

printf("Can't find the value\n");

}

return first;

}

NODE delete\_value(NODE first, int key) {

NODE current = first;

if (first == NULL) {

printf("Can't delete from empty list\n");

return NULL;

}

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;

}

if (current->next != NULL) {

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

}

free(current);

} else {

printf("Value not found\n");

}

return first;

}

void display(NODE first) {

NODE current = first;

while (current != NULL) {

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

current = current->next;

}

printf("NULL\n");

}

void main() {

NODE head = NULL;

int ch, val, key;

while (1) {

printf("Enter choice:\n1. Insert at left of value\n2. Delete value\n3. Display\n4. Exit\n");

scanf("%d", &ch);

switch (ch) {

case 1:

printf("Enter the item and key: ");

scanf("%d%d", &val, &key);

head = insert\_left\_value(head, val, key);

break;

case 2:

printf("Enter value to delete: ");

scanf("%d", &val);

head = delete\_value(head, val);

break;

case 3:

display(head);

break;

case 4:

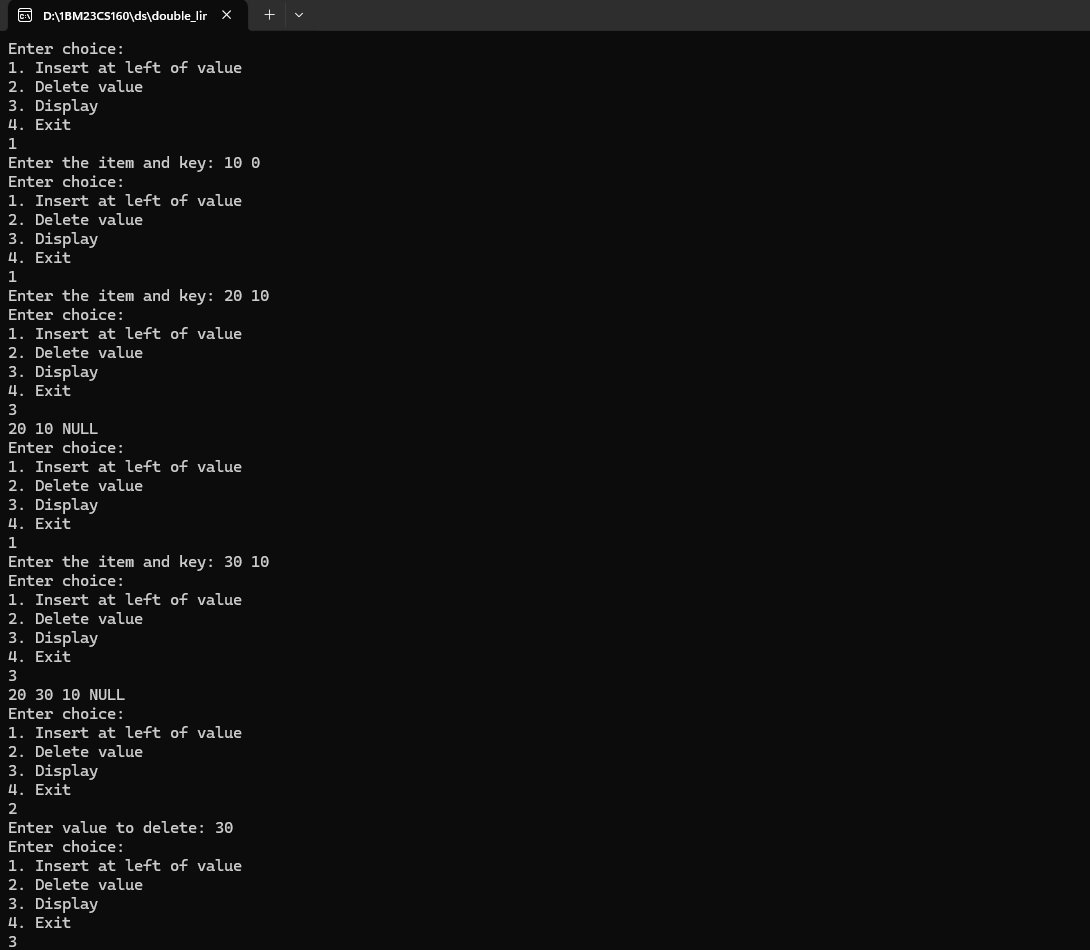
exit(0);

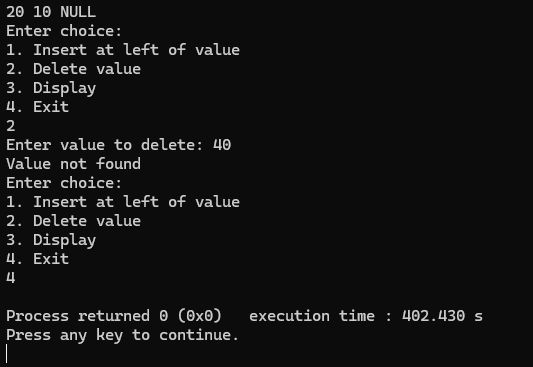
}

}

}

**Output:**





**LeetCode 3:**

**Middle of the Linked List**

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

struct ListNode\* current = head;

int c=1,m;

if(head==NULL)

return NULL;

while(current->next!=NULL){

current=current->next;

c++;

}

m=c/2;

current=head;

c=0;

while(c!=m && current->next!=NULL){

current=current->next;

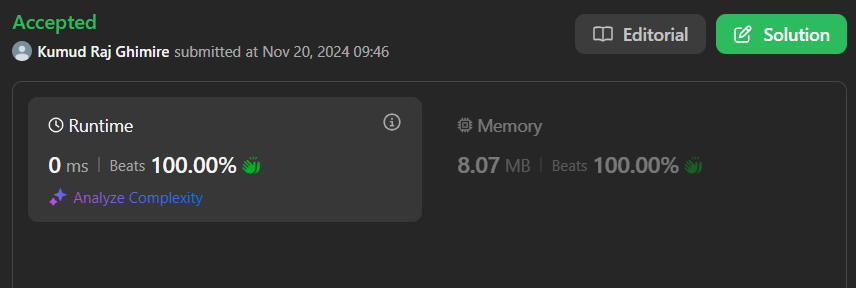
c++;

}

return current;

}

**Output:**

****

**Lab program 8:**

**Write a program a) To construct a binary Search tree. b) To traverse the tree using all the methods i.e., in-order, preorder and post order c) To display the elements in the tree.**

#include<stdio.h>

#include<stdlib.h>

struct BST{

int value;

struct BST\* left;

struct BST\* right;

};

typedef struct BST node;

node\* create(){

node \*temp = (node\*)malloc(sizeof(node));

printf("Enter the value: ");

scanf("%d",&(temp->value));

temp->left=NULL;

temp->right=NULL;

return temp;

}

void insert(node\* root, node\* temp){

if(temp->value>root->value){

if(root->right!=NULL){

insert(root->right,temp);

}

else{

root->right=temp;

}

}

if(temp->value<root->value){

if(root->left!=NULL){

insert(root->left,temp);

}

else{

root->left=temp;

}

}

}

void preorder(node \*root){

if(root!=NULL){

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

preorder(root->left);

preorder(root->right);

}

}

void inorder(node \*root){

if(root!=NULL){

preorder(root->left);

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

preorder(root->right);

}

}

void postorder(node \*root){

if(root!=NULL){

preorder(root->left);

preorder(root->right);

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

}

}

int main(){

int n;

node \*root = NULL, \*temp;

printf("How many value's do you want to enter: ");

scanf("%d", &n);

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

temp = create();

if(root==NULL){

root=temp;

}

else

{

insert(root,temp);

}

}

printf("\npre traval: \n");

preorder(root);

printf("\nin traval: \n");

inorder(root);

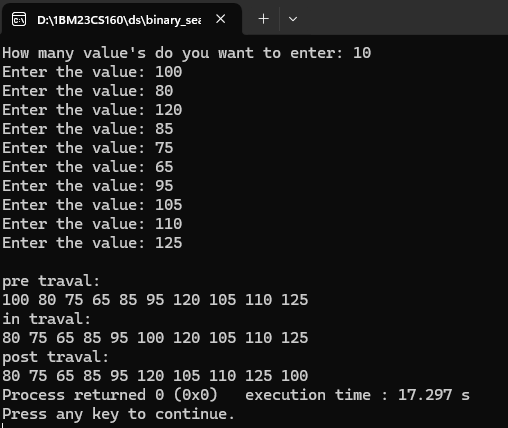
printf("\npost traval: \n");

postorder(root);

return 0;

}

**Output:**

****

**Lab Program 9a:**

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

#include<stdio.h>

int a[10][10], vis[10], n;

void bfs(int v){

int q[10], f=1, r=1, u,i;

q[r]=v;

vis[v]=1;

char s[5] = {'A','B','C','D','E'};

while(f<=r){

u=q[f];

printf("%c ",s[u]);

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

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

vis[i]=1;

r++;

q[r]=i;

}

}

f=f+1;

}

}

void main(){

int i,j,src;

printf("Enter the number of vertices \n");

scanf("%d",&n);

printf("Enter the adjacency matrix \n");

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

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

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

}

vis[i]=0;

}

printf("Enter the src vertex \n");

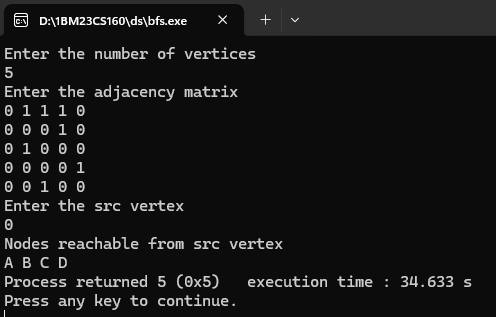
scanf("%d",&src);

printf("Nodes reachable from src vertex \n");

bfs(src);

}

**Output:**

****

**Lab Program 9b:**

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

#include<stdio.h>

void dfs(int, int \*, int \*);

int n, i, j, a[10][10], vis[10];

int isConnected = 1, isCyclic = 0;

void main() {

printf("Enter the number of vertices\n");

scanf("%d", &n);

printf("Enter the adjacency matrix\n");

for (i = 1; i <= n; i++) {

for (j = 1; j <= n; j++) {

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

}

vis[i] = 0;

}

for (i = 1; i <= n; i++) {

if (vis[i] == 0)

dfs(i, &isConnected, &isCyclic);

}

for (i = 1; i <= n; i++) {

if (vis[i] == 0) {

isConnected = 0;

break;

}

}

if (isConnected)

printf("\nThe graph is connected");

else

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

if (isCyclic)

printf("\nThe graph is cyclic\n");

else

printf("\nThe graph is not cyclic\n");

}

void dfs(int v, int \*isConnected, int \*isCyclic) {

vis[v] = 1;

char s[10] = {' ', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I'};

printf("%c ", s[v]);

for (j = 1; j <= n; j++) {

if (a[v][j] == 1) {

if (vis[j] == 0) {

dfs(j, isConnected, isCyclic);

} else if (vis[j] == 1) {

\*isCyclic = 1;

}

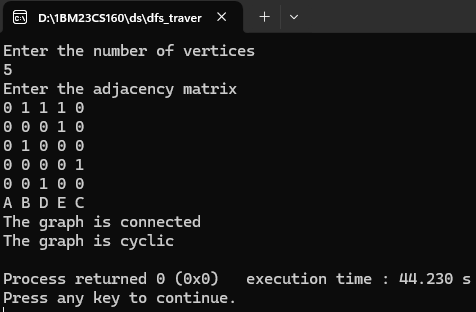
}

}

vis[v] = 2;

}

**Output:**

****