**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

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

****

**LAB REPORT**

**On**

**DATA STRUCTURES (23CS3PCDST)**

**Submitted by**

**RAKSHIT S BHAT (1BM23CS261)**

**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 RAKSHIT S BHAT **(1BM23CS261)**, 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 | 6-10 |
| 3 | Working of queue | 10-14 |
| 4 | Singly linked list -insertion | 14-24 |
| 5 | Singly linked list-deletion | 14-24 |
| 6 | sort ,reverse and concatenate;implementing stack and queue | 25-35 |
| 7 | Doubly linked list | 35-42 |
| 8 | Binary | 42-45 |
| 9 | BFS traversal;DFS traversal | 45-51 |
| 10 | Linear probing | 52-55 |

**Course outcomes:**

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

**Lab program 1:**

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

**a) Push**

**b) Pop**

**c) Display**

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

#include <stdio.h>

#include <stdlib.h>

#define SIZE 5

int top = -1;

char stack[SIZE];

void push(char a);

void pop();

void display();

int main()

{

int choice;

char ele;

do

{

printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");

scanf("%d", &choice);

getchar();

switch(choice)

{

case 1:

printf("Enter the char to push: ");

scanf("%c", &ele);

getchar();

push(ele);

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

printf("Exiting the program\n");

break;

default:

printf("Invalid\n");

break;

}

}while (choice != 4);

}

void push(char a)

{

if (top == SIZE - 1)

{

printf("Stack overflow\n");

return;

}

stack[++top] = a;

}

void pop()

{

if (top == -1)

{

printf("Stack underflow\n");

return;

}

printf("The popped element is : %c\n", stack[top--]);

return;

}

void display()

{

if (top == -1)

{

printf("Empty\n");

return;

}

printf("The stack elements are\n");

for (int i = top; i >= 0; i--)

{

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

}

return;

}

**Output:**



**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 <stdlib.h>

#include <string.h>

int index1 = 0, pos = 0, top = -1, length;

char symbol, temp, infix[20], postfix[20], stack[20];

void infixtopostfix();

int pred(char symbol);

char pop();

void push(char a);

int main()

{

printf("Enter the infix expression: ");

fgets(infix, sizeof(infix), stdin);

infix[strcspn(infix, "\n")] = '\0';

infixtopostfix();

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

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

}

void infixtopostfix()

{

length = strlen(infix);

push('#');

while (index1 < length)

{

symbol = infix[index1];

switch (symbol)

{

case '(':;

push(symbol);

break;

case ')':

temp = pop();

while (temp != '(')

{

postfix[pos] = temp;

pos++;

temp = pop();

}

break;

case '+':

case '-':

case '\*':

case '/':

case '^':

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

{

temp = pop();

postfix[pos++] = temp;

}

push(symbol);

break;

default:

postfix[pos++] = symbol;

break;

}

index1++;

}

while (top > 0)

{

temp = pop();

postfix[pos++] = temp;

}

postfix[pos] = '\0';

}

void push(char symbol)

{

top++;

stack[top] = symbol;

}

char pop()

{

char symb;

symb = stack[top];

top = top - 1;

return symb;

}

int pred(char symbol)

{

int p;

switch(symbol)

{

case '^':

p = 3;

break;

case '\*':

case '/':

p = 2;

break;

case '+':

case '-':

p = 1;

break;

case '(':

p = 0;

break;

case '#':

p = -1;

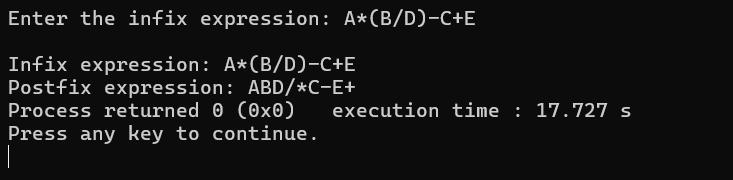
break;

}

return p;

}

**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>

#include <conio.h>

#define SIZE 3

int q[SIZE];

int front = -1; rear = -1;

void insert(void);

void delete(void);

void display(void);

int main()

{

int opt, val;

do

{

printf("1.Insert\n2.Delete\n3.Display\n4.Exit\n");

printf("Enter your choice: ");

scanf("%d", &opt);

getchar();

switch(opt)

{

case 1:

insert();

break;

case 2:

delete();

break;

case 3:

display();

break;

case 4:

printf("Exiting\n");

break;

default:

printf("Invalid\n");

break;

}

} while (opt != 4);

}

void insert()

{

if (rear == SIZE - 1)

{

printf("Overflow\n");

return;

}

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

front = rear = 0;

else

rear++;

int num;

printf("Enter the number to be inserted: ");

scanf("%d", &num);

getchar();

q[rear] = num;

}

void delete()

{

int val;

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

{

printf("Underflow\n");

return;

}

val = q[front++];

printf("The number deleted is %d\n", val);

if (front > rear)

front = rear = -1;

return;

}

void display()

{

int i;

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

{

printf("Queue is empty\n");

return;

}

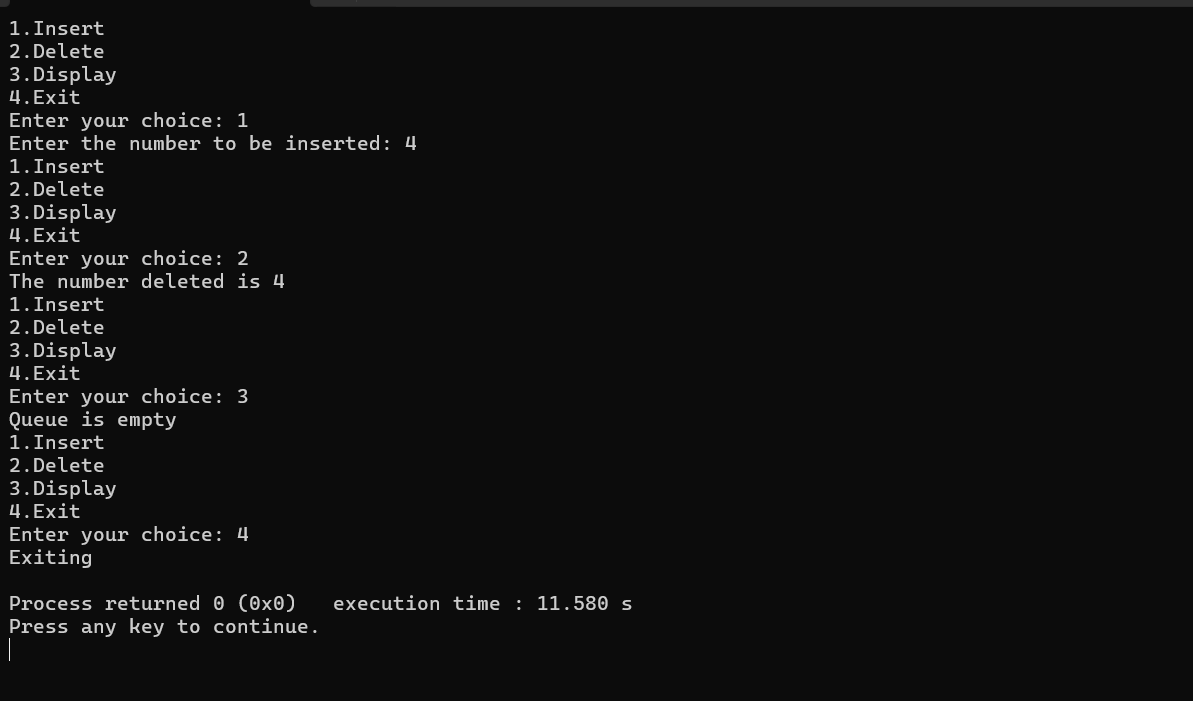
for (i = front; i <= rear; i++)

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

printf("\n");

}

**OUTPUT:**

****

**Lab program 4:**

**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.**

**LAB PROGRAM 5:**

**Implement Singly Linked List with following operations  
a)   Create a linked list.  
b)   Deletion of first element, specified element and last element in the list.  
c) Display the contents of the linked list.**

#include <stdio.h>

#include <stdlib.h>

typedef struct node {

int data;

struct node \*next;

} node;

void display(node \*start);

node \*create\_node(int num);

node \*insert\_beg(node \*\*start);

node \*insert\_end(node \*start);

node \*insert\_pos(node \*\*start);

node \*delete\_beg(node \*\*start);

node \*delete\_end(node \*\*start);

node \*delete\_pos(node \*\*start);

int main()

{

int choice;

node \*start = NULL;

do

{

printf("\n1. Insert at beginning\n2. Insert at end\n3. Insert at position\n4. Delete at beginning\n5. Delete at position\n6. Delete at end\n7. Display list\n8. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch(choice)

{

case 1:

start = insert\_beg(&start);

break;

case 2:

start = insert\_end(start);

break;

case 3:

start = insert\_pos(&start);

break;

case 4:

start = delete\_beg(&start);

break;

case 5:

start = delete\_pos(&start);

break;

case 6:

start = delete\_end(&start);

break;

case 7:

display(start);

break;

case 8:

printf("Exiting program\n");

break;

default:

printf("Invalid choice\n");

break;

}

} while (choice != 8);

}

node \*create\_node(int num)

{

node \*new\_node;

new\_node = (node\*) malloc(sizeof(node));

if (!new\_node)

{

printf("Memory allocation failed\n");

exit(1);

}

new\_node->data = num;

new\_node->next = NULL;

return new\_node;

}

node \*insert\_beg(node \*\*start)

{

node \*new\_node;

int val;

printf("Enter data to insert at beg: ");

scanf("%d", &val);

getchar();

new\_node = create\_node(val);

if (\*start == NULL)

return new\_node;

new\_node->next = \*start;

\*start = new\_node;

return \*start;

}

node \*insert\_end(node \*start)

{

node \*new\_node, \*ptr;

int val;

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

scanf("%d", &val);

getchar();

new\_node = create\_node(val);

if (start == NULL)

return new\_node;

ptr = start;

while (ptr->next != NULL)

ptr = ptr->next;

ptr->next = new\_node;

return start;

}

node \*insert\_pos(node \*\*start)

{

node \*new\_node, \*ptr;

int val, pos, count = 1;

ptr = \*start;

while (ptr != NULL)

{

ptr = ptr->next;

count++;

}

printf("Enter data to insert: ");

scanf("%d", &val);

getchar();

new\_node = create\_node(val);

do {

printf("Enter position: ");

scanf("%d", &pos);

if (pos > count || pos <= 0)

printf("Unable to insert at that position\n");

} while (pos > count || pos <= 0);

ptr = \*start;

for (int i = 0; i < pos - 2; i++)

ptr = ptr->next;

if (pos == 1)

{

new\_node->next = \*start;

\*start = new\_node;

}

else

{

new\_node->next = ptr->next;

ptr->next = new\_node;

}

return \*start;

}

node \*delete\_beg(node \*\*start)

{

if (\*start == NULL)

{

printf("List is empty\n");

return NULL;

}

node \*ptr = \*start;

\*start = ptr->next;

free(ptr);

return \*start;

}

node \*delete\_end(node \*\*start)

{

if (\*start == NULL)

{

printf("List is empty\n");

return NULL;

}

node \*ptr, \*preptr;

ptr = \*start;

if (ptr->next == NULL)

{

\*start = NULL;

free(ptr);

printf("Element deleted\n");

return \*start;

}

while (ptr->next != NULL)

{

preptr = ptr;

ptr = ptr->next;

}

free(ptr);

preptr->next = NULL;

printf("Element deleted\n");

return \*start;

}

node \*delete\_pos(node \*\*start)

{

if (\*start == NULL)

{

printf("List is empty\n");

return NULL;

}

node \*ptr, \*preptr;

int pos, count = 1;

ptr = \*start;

while (ptr->next != NULL)

{

ptr = ptr->next;

count++;

}

do {

printf("Enter position: ");

scanf("%d", &pos);

if (pos > count || pos <= 0)

printf("Unable to insert at that position\n");

} while (pos > count || pos <= 0);

ptr = \*start;

if (pos == 1)

{

\*start = ptr->next;

free(ptr);

printf("Element deleted\n");

return \*start;

}

for (int i = 0; i < pos - 1; i++)

{

preptr = ptr;

ptr = ptr->next;

}

preptr->next = ptr->next;

free(ptr);

printf("Element deleted\n");

return \*start;

}

void display(node \*start)

{

if (start == NULL)

{

printf("List is empty\n");

return;

}

node \*ptr = start;

while (ptr != NULL)

{

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

ptr = ptr->next;

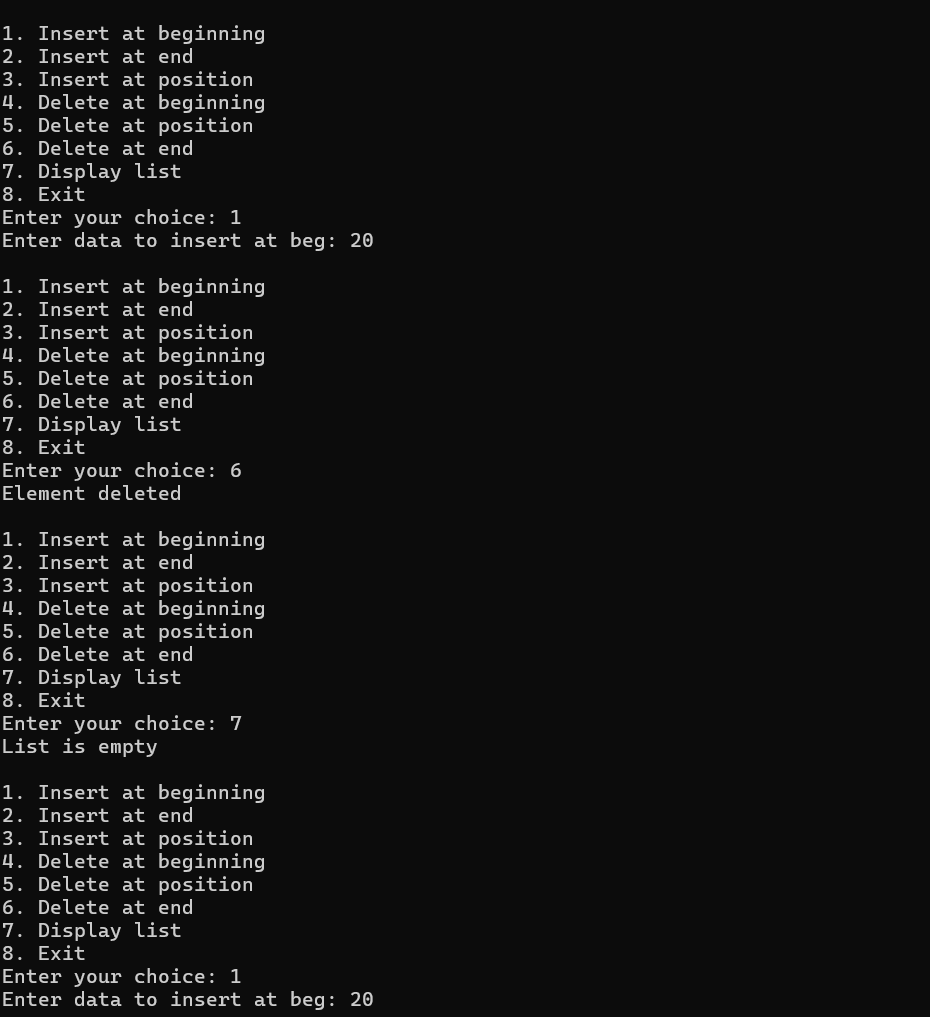
}

printf("NULL\n");

return;

}

**OUTPUT:**

****

****

**A screen shot of a computer

Description automatically generated**

**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**

#include <stdio.h>

#include <stdlib.h>

typedef struct node

{

int data;

struct node \*next;

} node;

node \*create\_ll(void);

void reverse(node \*\*start);

void concat(node \*start1, node \*start2);

void sort(node \*start);

void append(node \*start, int val);

int main()

{

node \*start1 = create\_ll();

node \*start2 = create\_ll();

reverse(&start1);

sort(start2);

concat(start1, start2);

printf("Final concatenated list:\n");

node \*ptr = start1;

while (ptr != NULL)

{

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

ptr = ptr->next;

}

printf("NULL\n");

return 0;

}

node \*create\_ll(void)

{

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

if (!start)

{

printf("Memory allocation error\n");

exit(1);

}

start->data = 10;

start->next = NULL;

append(start, 15);

append(start, 20);

append(start, 9);

printf("Initial linked list:\n");

node \*ptr = start;

while (ptr != NULL)

{

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

ptr = ptr->next;

}

printf("NULL\n");

return start;

}

void reverse(node \*\*start)

{

node \*prev = NULL, \*current = \*start, \*next = NULL;

while (current != NULL)

{

next = current->next;

current->next = prev;

prev = current;

current = next;

}

\*start = prev;

printf("List after reversing:\n");

node \*ptr = \*start;

while (ptr != NULL)

{

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

ptr = ptr->next;

}

printf("NULL\n");

}

void concat(node \*start1, node \*start2)

{

node \*ptr = start1;

while (ptr->next != NULL)

{

ptr = ptr->next;

}

ptr->next = start2;

}

void sort(node \*start)

{

if (!start) return;

node \*ptr;

int temp, swapped;

do

{

swapped = 0;

ptr = start;

while (ptr->next != NULL)

{

if (ptr->data > ptr->next->data)

{

temp = ptr->data;

ptr->data = ptr->next->data;

ptr->next->data = temp;

swapped = 1;

}

ptr = ptr->next;

}

} while (swapped);

printf("List after sorting:\n");

ptr = start;

while (ptr != NULL)

{

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

ptr = ptr->next;

}

printf("NULL\n");

}

void append(node \*start, int val)

{

node \*new\_node = (node \*)malloc(sizeof(node));

if (!new\_node)

{

printf("Memory allocation error\n");

return;

}

new\_node->data = val;

new\_node->next = NULL;

node \*ptr = start;

while (ptr->next != NULL)

{

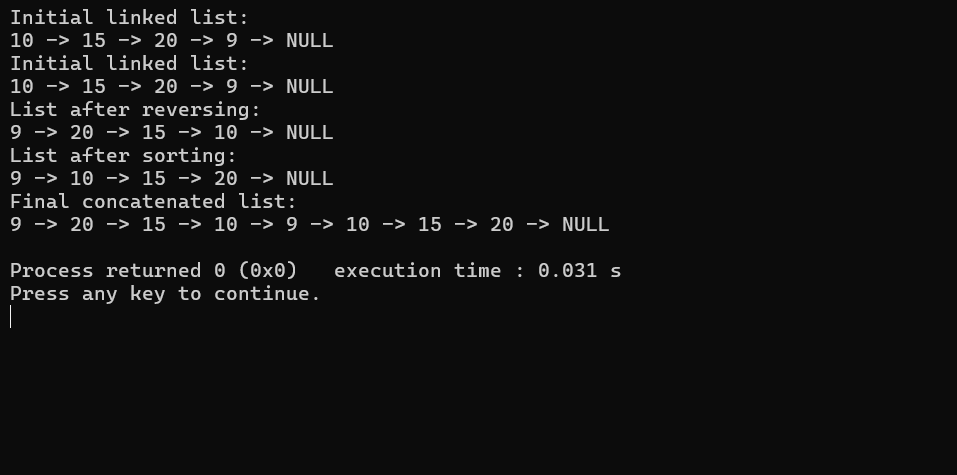
ptr = ptr->next;

}

ptr->next = new\_node;

}

**OUTPUT:**

****

**6b) WAP to Implement Single Link List  
to simulate Stack & Queue  
Operations. (stack and queue to be implemented as one program)**

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \*next;

};

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

struct node \*head = NULL;

void enqueue(int val) {

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

newNode->data = val;

newNode->next = NULL;

if (front == NULL && rear == NULL) {

front = rear = newNode;

} else {

rear->next = newNode;

rear = newNode;

}

}

void dequeue() {

struct node \*temp;

if (front == NULL) {

printf("Queue is Empty. Unable to perform dequeue.\n");

} else {

temp = front;

front = front->next;

if (front == NULL) {

rear = NULL;

}

printf("Dequeued element = %d\n", temp->data);

free(temp);

}

}

void printQueue() {

struct node \*temp = front;

printf("Queue: ");

while (temp) {

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

temp = temp->next;

}

printf("NULL\n");

}

void push(int val) {

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

newNode->data = val;

newNode->next = head;

head = newNode;

}

void pop() {

struct node \*temp;

if (head == NULL) {

printf("Stack is Empty.\n");

} else {

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

temp = head;

head = head->next;

free(temp);

}

}

void printStack() {

struct node \*temp = head;

printf("Stack: ");

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

int main() {

int choice, value;

do {

printf("\nSelect an operation:\n");

printf("1. Enqueue \n");

printf("2. Dequeue \n");

printf("3. Push \n");

printf("4. Pop \n");

printf("5. Queue\n");

printf("6. Stack\n");

printf("0. Exit\n");

printf("Enter choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to enqueue: ");

scanf("%d", &value);

enqueue(value);

break;

case 2:

dequeue();

break;

case 3:

printf("Enter value to push: ");

scanf("%d", &value);

push(value);

break;

case 4:

pop();

break;

case 5:

printQueue();

break;

case 6:

printStack();

break;

case 0:

printf("Exiting program.\n");

break;

default:

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

}

} while (choice != 0);

return 0;

}

**OUTPUT:**

**A screenshot of a computer program

Description automatically generated A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**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>

typedef struct node {

int data;

struct node \*next;

struct node \*prev;

} node;

node \*create\_ll(void);

void insert\_left(node \*\*head, int target, int val);

void delete\_node(node \*\*head, int target);

void display(node \*head);

int main() {

int choice = 0, val, target;

node \*head = NULL;

while (choice != 5) {

printf("\nEnter your choice:\n");

printf("1. Create ll\n2. Display ll\n3. Insert to left of a node\n4. Delete a node\n5. Exit\n");

scanf("%d", &choice);

switch (choice) {

case 1:

head = create\_ll();

break;

case 2:

display(head);

break;

case 3:

printf("Enter value of target node: ");

scanf("%d", &target);

printf("Enter value for new node: ");

scanf("%d", &val);

insert\_left(&head, target, val);

break;

case 4:

printf("Enter value of node to delete: ");

scanf("%d", &target);

delete\_node(&head, target);

break;

case 5:

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

break;

default:

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

break;

}

}

while (head != NULL) {

node \*temp = head;

head = head->next;

free(temp);

}

return 0;

}

node \*create\_ll(void) {

int choice = 0;

node \*head = NULL, \*ptr = NULL, \*new\_node = NULL;

while (choice != 2) {

printf("1. Add Node\n2. Exit\n");

scanf("%d", &choice);

if (choice == 2) {

break;

}

new\_node = (node \*)malloc(sizeof(node));

if (!new\_node) {

printf("Memory allocation error\n");

exit(1);

}

printf("Enter value for new node: ");

scanf("%d", &new\_node->data);

new\_node->prev = NULL;

new\_node->next = NULL;

if (head == NULL) {

head = new\_node;

} else {

ptr->next = new\_node;

new\_node->prev = ptr;

}

ptr = new\_node;

}

return head;

}

void insert\_left(node \*\*head, int target, int val) {

node \*temp, \*new\_node;

if (!(\*head)) {

printf("List is empty. Cannot insert.\n");

return;

}

temp = \*head;

while (temp != NULL) {

if (temp->data == target) {

new\_node = (node \*)malloc(sizeof(node));

if (!new\_node) {

printf("Memory allocation error\n");

exit(2);

}

new\_node->data = val;

new\_node->next = temp;

new\_node->prev = temp->prev;

if (temp == \*head) {

\*head = new\_node;

} else {

temp->prev->next = new\_node;

}

temp->prev = new\_node;

return;

}

temp = temp->next;

}

printf("Target node not found.\n");

}

void delete\_node(node \*\*head, int target) {

if (!(\*head)) {

printf("List is empty. Cannot delete.\n");

return;

}

node \*ptr = \*head;

while (ptr != NULL) {

if (ptr->data == target) {

if (ptr == \*head) {

\*head = ptr->next;

if (\*head != NULL) {

(\*head)->prev = NULL;

}

} else if (ptr->next == NULL) {

ptr->prev->next = NULL;

} else {

ptr->prev->next = ptr->next;

ptr->next->prev = ptr->prev;

}

free(ptr);

printf("Node with value %d deleted successfully.\n", target);

return;

}

ptr = ptr->next;

}

printf("Target node not found.\n");

}

void display(node \*head) {

if (!head) {

printf("List is empty.\n");

return;

}

node \*temp = head;

printf("Linked list: ");

while (temp != NULL) {

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

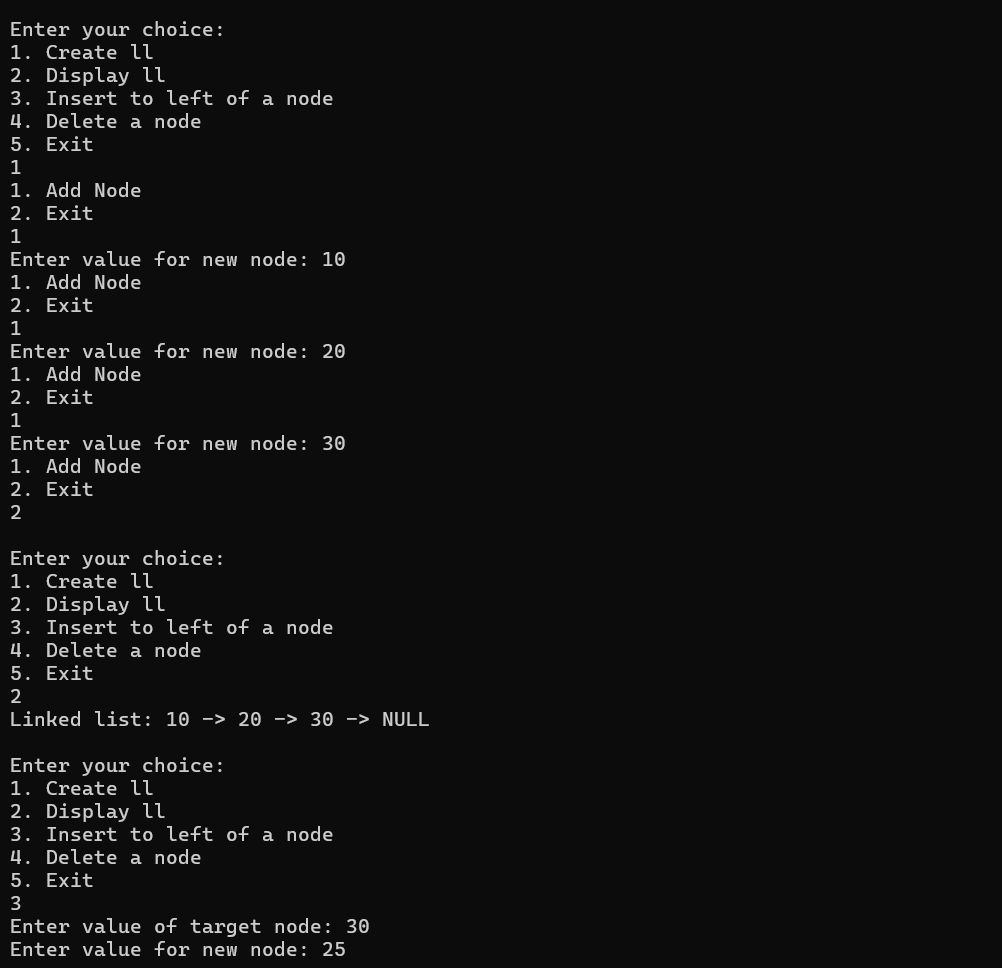
temp = temp->next;

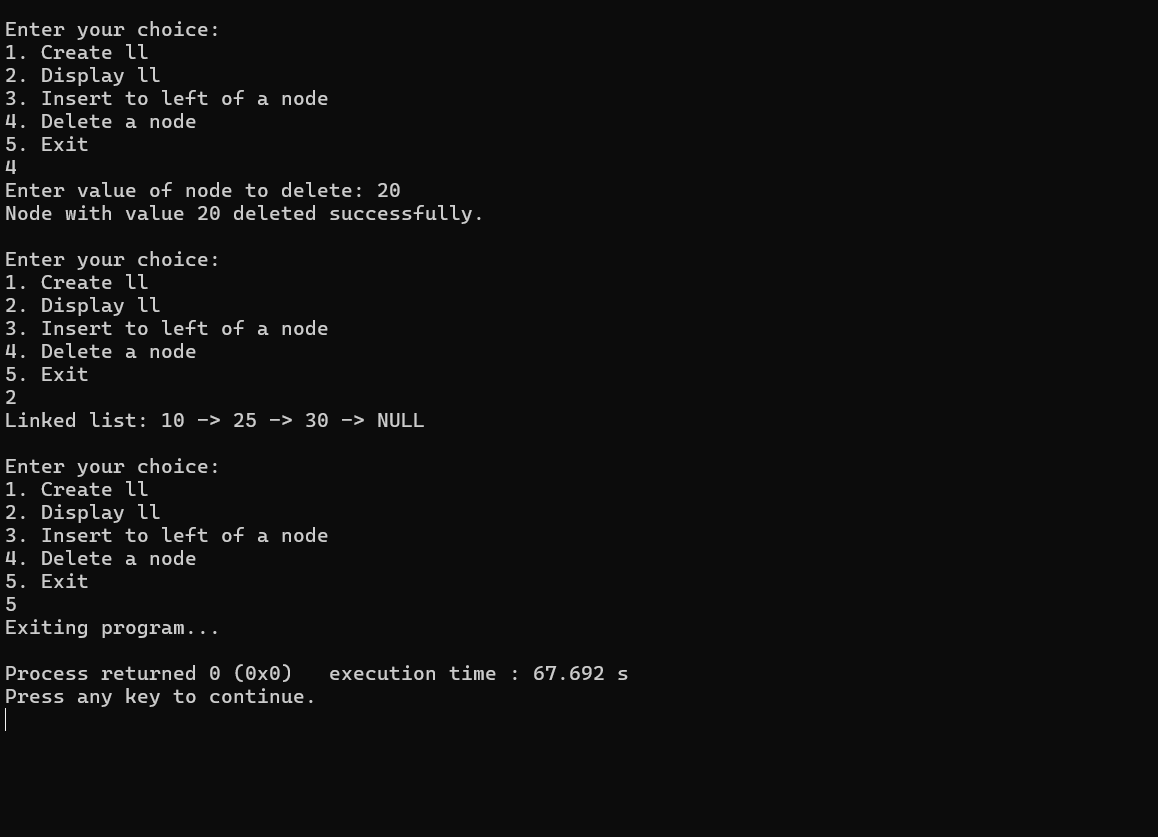
}

printf("NULL\n");

}

**OUTPUT:**

****

****

**LAB PROGRAM 8:**

**Binary search tree**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int value;

struct Node\* left;

struct Node\* right;

};

struct Node\* createNode(int key) {

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

newNode->value = key;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

struct Node\* insert(struct Node\* root, int key) {

if (root == NULL) {

return createNode(key);

}

if (key < root->value) {

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

} else {

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

}

return root;

}

void inorder(struct Node\* root) {

if (root != NULL) {

inorder(root->left);

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

inorder(root->right);

}

}

void preorder(struct Node\* root) {

if (root != NULL) {

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

preorder(root->left);

preorder(root->right);

}

}

void postorder(struct Node\* root) {

if (root != NULL) {

postorder(root->left);

postorder(root->right);

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

}

}

int main() {

struct Node\* root = NULL;

int n, value;

printf("Enter the number of elements to insert into the BST: ");

scanf("%d", &n);

printf("Enter the values to insert into the BST:\n");

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

scanf("%d", &value);

root = insert(root, value);

}

printf("\nIn-order Traversal: ");

inorder(root);

printf("\n");

printf("Pre-order Traversal: ");

preorder(root);

printf("\n");

printf("Post-order Traversal: ");

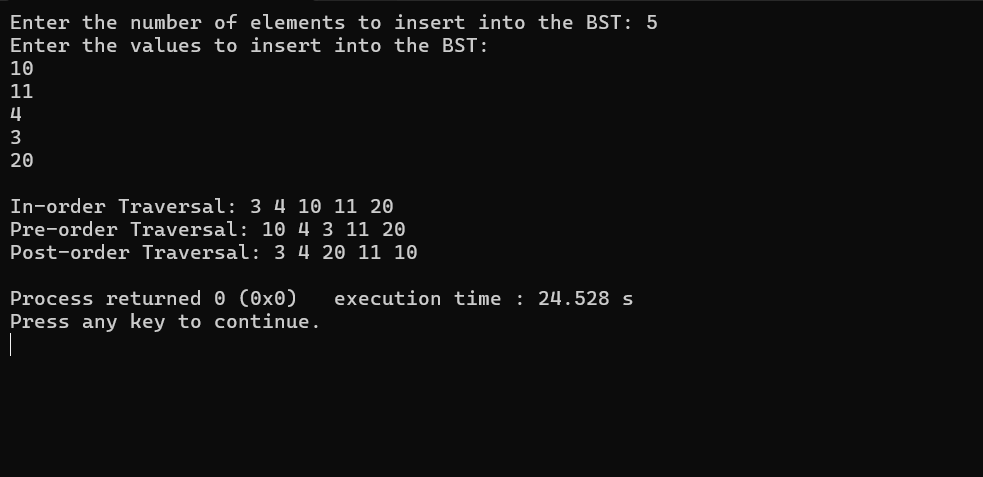
postorder(root);

printf("\n");

return 0;

}

**OUTPUT:**

****

**LAB PROGRAM 9:**

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

#include <stdio.h>

#include <stdlib.h>

#define MAX\_VERTICES 100

struct Queue {

int items[MAX\_VERTICES];

int front, rear;

};

void initQueue(struct Queue\* q) {

q->front = -1;

q->rear = -1;

}

int isQueueEmpty(struct Queue\* q) {

return (q->front == -1);

}

int isQueueFull(struct Queue\* q) {

return (q->rear == MAX\_VERTICES - 1);

}

void enqueue(struct Queue\* q, int value) {

if (isQueueFull(q)) {

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

return;

}

if (q->front == -1)

q->front = 0;

q->rear++;

q->items[q->rear] = value;

}

int dequeue(struct Queue\* q) {

if (isQueueEmpty(q)) {

printf("Queue is empty!\n");

return -1;

}

int item = q->items[q->front];

q->front++;

if (q->front > q->rear) {

q->front = q->rear = -1;

}

return item;

}

void bfs(int graph[MAX\_VERTICES][MAX\_VERTICES], int startVertex, int n) {

struct Queue q;

initQueue(&q);

int visited[MAX\_VERTICES] = {0};

enqueue(&q, startVertex);

visited[startVertex] = 1;

printf("BFS Traversal starting from vertex %d: ", startVertex);

while (!isQueueEmpty(&q)) {

int currentVertex = dequeue(&q);

printf("%d ", currentVertex);

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

if (graph[currentVertex][i] == 1 && !visited[i]) {

enqueue(&q, i);

visited[i] = 1;

}

}

}

printf("\n");

}

int main() {

int graph[MAX\_VERTICES][MAX\_VERTICES] = {0};

int n, m, u, v, startVertex;

printf("Enter the number of vertices: ");

scanf("%d", &n);

printf("Enter the number of edges: ");

scanf("%d", &m);

printf("Enter the edges (u v) representing an undirected graph:\n");

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

scanf("%d %d", &u, &v);

graph[u][v] = 1;

graph[v][u] = 1;

}

printf("Enter the starting vertex for BFS: ");

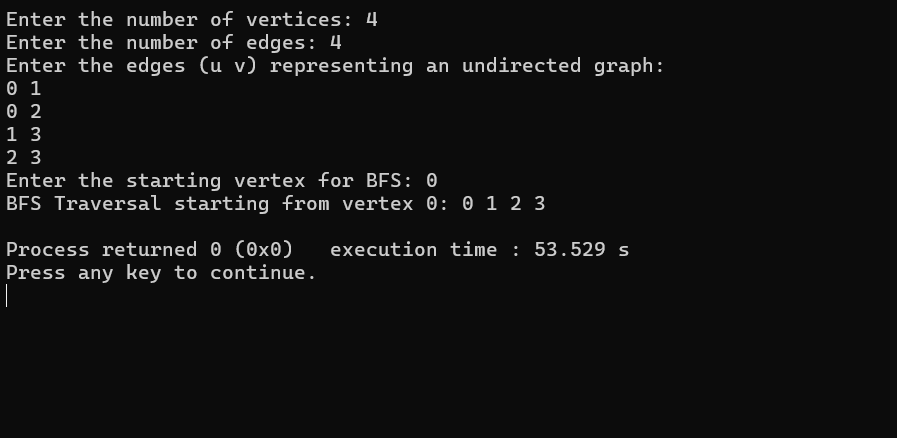
scanf("%d", &startVertex);

bfs(graph, startVertex, n);

return 0;

}

**OUTPUT:**

****

**LAB PROGRAM**

**9B) DFS TRAVERSAL**

#include <stdio.h>

#define MAX 5\

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

int stack[MAX], top = -1, i;\

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

visited[k] = 0;\

stack[++top] = start;

visited[start] = 1;\

while (top != -1) {

start = stack[top--];

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

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

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

stack[++top] = 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]);\

// Perform DFS from the first node (0)

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

dfs(adj, visited, 0);

return 0;

}

**OUTPUT:**

**A screenshot of a computer program

Description automatically generated**

**LAB PROGRAM 10:**

**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:**

**A screenshot of a computer program

Description automatically generated**

**A black screen with white text

Description automatically generated**

**LeetCode Problems**

**1. Move Zeroes**

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

int tmp, \*j;

int \*ptr = nums;

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

{

if (\*ptr != 0)

ptr++;

else

{

j = ptr;

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

{

tmp = \*j;

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

\*(j+1) = tmp;

j++;

}

}

}

}

**Output:**

A screenshot of a test

Description automatically generated

**2. Majority Element**

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

int ele;

int count;

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

{

count = 0;

if(nums[i]==ele){

continue;

}

ele = nums[i];

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

{

if(nums[j]==ele)

count++;

}

if (count>(numsSize/2))

{

return ele;

break;

}

}

return ele;

}

**Output:**

**A screenshot of a computer

Description automatically generated**

**3. Linked list palindrome**

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;

}

**Output:**

**A black and grey striped background

Description automatically generated**

**4. Path Sum**

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:**

**A screenshot of a computer

Description automatically generated**

**5. Hackerrank – Game of two stacks**

#include <math.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <assert.h>

#include <limits.h>

#include <stdbool.h>

int main(){

int g;

scanf("%d",&g);

for(int a0 = 0; a0 < g; a0++){

int n;

int m;

int x;

scanf("%d %d %d",&n,&m,&x);

int \*a = malloc(sizeof(int) \* n);

for(int a\_i = 0; a\_i < n; a\_i++){

scanf("%d",&a[a\_i]);

}

int \*b = malloc(sizeof(int) \* m);

for(int b\_i = 0; b\_i < m; b\_i++){

scanf("%d",&b[b\_i]);

}

// your code goes here

unsigned long long sum = 0;

int i = 0, j = 0;

while (i < n && sum+a[i] <= x) {

sum += a[i];

++i;

}

while (j < m && sum+b[j] <= x) {

sum += b[j];

++j;

}

int max = i+j;

while (i > 0) {

sum -= a[--i];

while (j < m && sum+b[j] <= x) {

sum += b[j];

++j;

}

if (i+j > max) {

max = i+j;

}

}

printf("%d\n", max);

}

return 0;

}

**Output:**

**A screenshot of a computer

Description automatically generated**