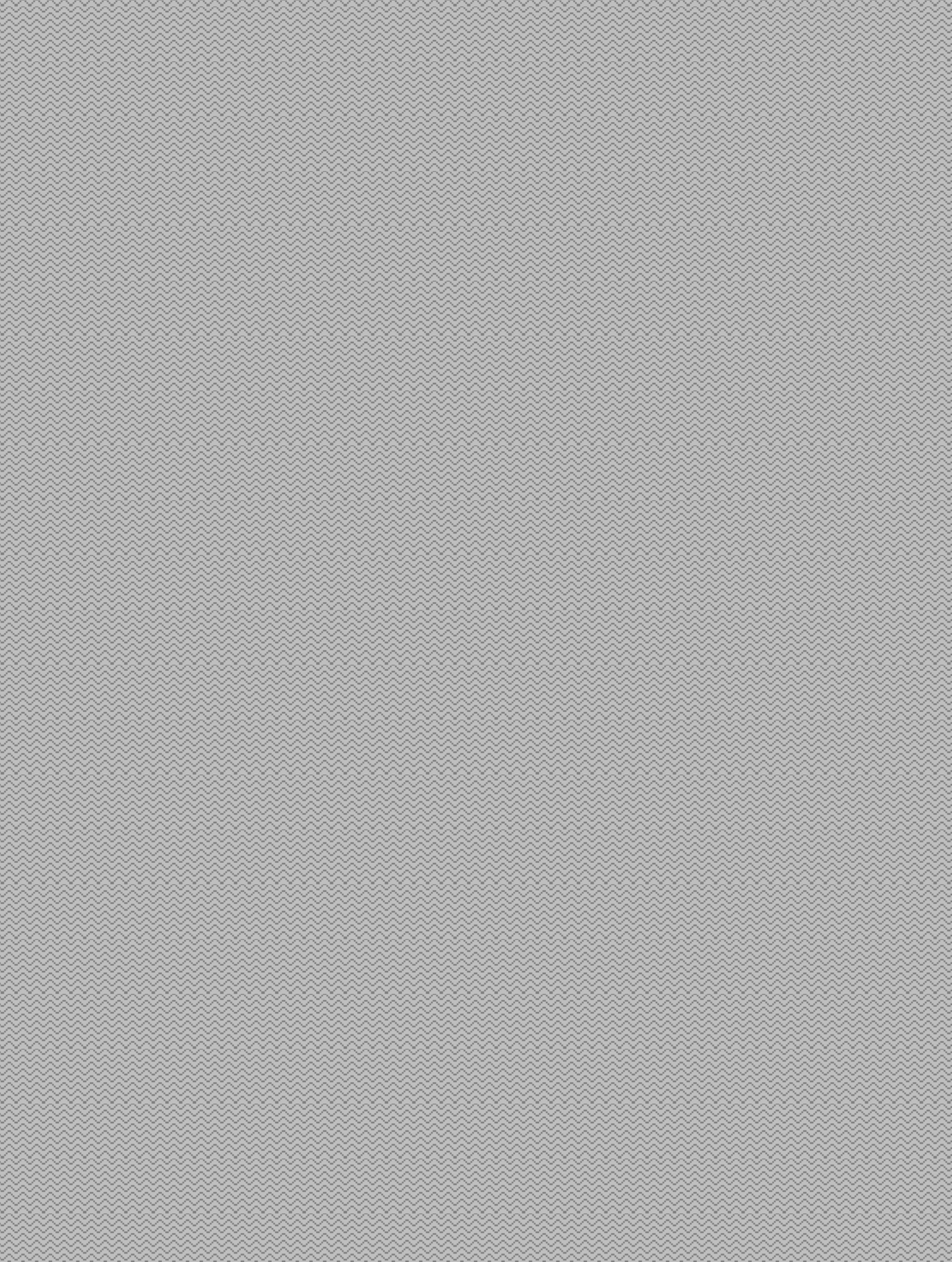
**Vadodar**

**Dat**



MAD

**(BE031000081)**

LABORATORY MANUAL

**B.E. Semester-III**

**Prepared By:- CE/IT Department**

**Vadodara Institute of Engineering (080) Kotambi, Vadodara – 391510**

**Academic Year : 2025-2026**

# Index

|  |  |
| --- | --- |
| **Sr No** | **Practicals** |
| **1** | Write android program to display "Welcome to android". |
| **2** | IWrite android program to demonstrate usage of string. |
| **3** | Write android program to demonstrate activity life cycle. |
| **4** | Write android program to change the background of activity. |
| **5** | Write android program to perform all operation in calculator. |
| **6** | Write a menu driven program to implement following operations on the singly linked list.   1. Insert a node at the front of the linked list. 2. Insert a node at the end of the linked list. 3. Insert a node such that linked list is in ascending order. (according to info. field)  1. Delete the first node of the linked list. 2. Delete a node with a given value in info. field. 3. Delete a node after specified position. |
| **7** | Write a program to implement stack using linked list. |
| **8** | Write a program to implement queue using linked list. |
| **9** | Write a program to implement following operations on the circular linked list. (a) Insert a node at the end of the linked list.   1. Insert a node before specified position. 2. Delete a first node of the linked list. 3. Delete a node after specified position. |
| **10** | Write a program to implement following operations on the doubly linked list. (a) Insert a node at the front of the linked list.   1. Insert a node at the end of the linked list. 2. Delete a last node of the linked list. 3. Delete a node before specified position. |
| **11** | Write a program which create binary search tree. Also write a function to search an element from binary search tree. |
| **12** | Write a program to implement Bubble Sort. |
| **13** | Write a program to implement Insertion Sort. |
| **14** | Write a program to create and display records using sequential file organization. |

**Software Requirements**

|  |  |  |
| --- | --- | --- |
| **Sr No** | **Software Requirement** | **Hardware Requirement** |
| **1** | Code::Blocks | 64-bit OS |
| **2** | Turbo C++ (Borland) | RAM-16 GB |
| **3** | Dev-C++ | Processor (Intel i3/i5/i7, AMD Ryzen, etc.) |
| **4** | Visual Studio (with C/C++ workload) | Hard Disk- 1 GB |

****

**Date:**

# Practical 1

**AIM: Write a program to demonstrate the concepts of Call by Value and Call by reference.**

## Call by value:

#include <stdio.h>

void swapx(int x, int y); int main()

{

int a = 10, b = 20;

swapx(a, b); // Actual Parameters printf("after swapping = %d b = %d\n", a, b); return 0;



= %d\n", x, y);

}

void swapx(int x, int y)

{

int t; t = x; x = y; y = t;

printf("before swapping= %d y

}

## Output :

before swapping= 20 y = 10 after swapping = 10 b = 20

**Call by Refrence:**

#include <stdio.h>

void swapx(int\*, int\*); int main()

{

int a = 10, b = 20;

swapx(&a, &b);

printf("after swapping = %d b = %d\n", a, b);

return 0;



}

void swapx(int\* x, int\* y)

{

int t;

t = \*x;

\*x = \*y;

\*y = t;

printf("before swapping= = %d y = %d\n", \*x, \*y);

}

## Output :

before swapping= 20 y = 10 after swapping = 10 b = 20

**Application:**

1. Write one difference between Call by value and Call by reference.
2. Write down synatx of call by Reference.

**Date:**

# Practical 2

**AIM: Implement a stack using arrays and perform the following operations: Push, Pop in c.**

## Program:

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

struct Stack { int top, cap; int \*a;

};

struct Stack\* createStack(int cap) {

struct Stack\* stack = (struct Stack\*)malloc(sizeof(struct Stack)); stack->cap = cap;

stack->top = -1;

stack->a = (int\*)malloc(cap \* sizeof(int)); return stack;

}

void deleteStack(struct Stack\* stack) { free(stack->a);

free(stack);

}

int isFull(struct Stack\* stack) {

return stack->top >= stack->cap - 1;

}

int isEmpty(struct Stack\* stack) { return stack->top < 0;

}

int push(struct Stack\* stack, int x) { if (isFull(stack)) {

printf("Stack Overflow\n"); return 0;

}

stack->a[++stack->top] = x;

return 1;

}

int pop(struct Stack\* stack) { if (isEmpty(stack)) {

printf("Stack Underflow\n"); return 0;

}

return stack->a[stack->top--];

}

int peek(struct Stack\* stack) { if (isEmpty(stack)) {

printf("Stack is Empty\n"); return 0;

}

return stack->a[stack->top];

}

int main() {

struct Stack\* s = createStack(5); push(s, 10);

push(s, 20);

push(s, 30);

printf("%d popped from stack\n", pop(s));

printf("Top element is: %d\n", peek(s));

printf("Elements present in stack: "); while (!isEmpty(s)) {

printf("%d ", peek(s)); pop(s);

}

deleteStack(s); return 0;

}

## Output :

30 popped from stack Top element is: 20

Elements present in stack: 20 10

**Application:**



1. What is stack?
2. What are the applications of stack?

**Date:**

# Practical 3

**AIM: Write a program to convert an infix expression to postfix using a stack.**

**Infix expression:** The expression of the form "a operator b" (a + b) i.e., when an operator is in- between every pair of operands.

**Postfix expression:** The expression of the form "a b operator" (ab+) i.e., When every pair of operands is followed by an operator.

## Examples:

**Input:** s = "A\*(B+C)/D"

**Output:** ABC+\*D/

## Program:

#include <stdio.h> #include <stdlib.h> #include <ctype.h> #include <string.h>

int prec(char c) { if (c == '^')

return 3;

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

else if (c == '+' || c == '-') return 1;

else

return -1;

}

void infixToPostfix(char\* exp) { int len = strlen(exp);

char result[len + 1]; char stack[len];

int j = 0;

int top = -1;

for (int i = 0; i < len; i++) { char c = exp[i];

// If the scanned character is

// an operand, add it to the output string.

if (isalnum(c)) result[j++] = c;

// If the scanned character is

// an ‘(‘, push it to the stack. else if (c == '(')

stack[++top] = '(';

// If the scanned character is an ‘)’,

// pop and add to the output string from the stack

// until an ‘(‘ is encountered. else if (c == ')') {

while (top != -1 && stack[top] != '(') { result[j++] = stack[top--];

}

top--;

}

// If an operator is scanned else {

while (top != -1 && (prec(c) < prec(stack[top]) || prec(c) == prec(stack[top]))) {

result[j++] = stack[top--];

}

stack[++top] = c;

}

}

// Pop all the remaining elements from the stack while (top != -1) {

result[j++] = stack[top--];

}

result[j] = '\0'; printf("%s\n", result);

}

int main() {

char exp[] = "a+b\*(c^d-e)^(f+g\*h)-i"; infixToPostfix(exp);

return 0;

}

## Output :

abcd^e-fgh\*+^\*+i-

**Application:**



1. Convert in postfix. ( A + B) \* C
2. How many notations are there in the polish notation.

**Date:**

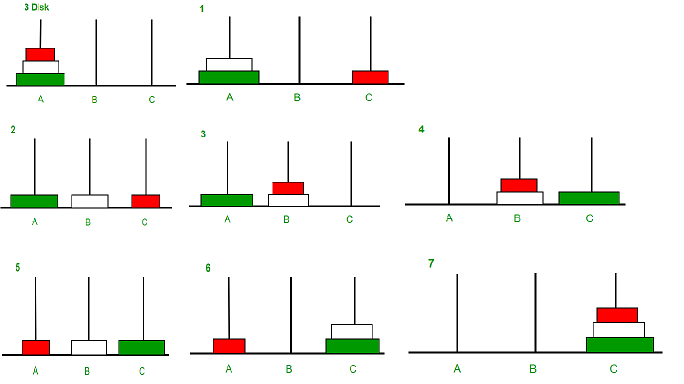
# Practical 4

**AIM: Implement the Tower of Hanoi problem using recursion.**

## Theory:

Tower of Hanoi is a mathematical puzzle where we have three rods (**A**, **B**, and **C**) and **N** disks. Initially, all the disks are stacked in decreasing value of diameter i.e., the smallest disk is placed on the top and they are on rod **A**. The objective of the puzzle is to move the entire stack to another rod (here considered **C**), obeying the following simple rules:

* + Only one disk can be moved at a time.
  + Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
  + No disk may be placed on top of a smaller disk.



## Tower of Hanoi using Recursion

The idea is to use the helper node to reach the destination using recursion. Below is the pattern for this problem:

* + Shift 'N-1' disks from 'A' to 'B', using C.
  + Shift last disk from 'A' to 'C'.
  + Shift 'N-1' disks from 'B' to 'C', using A.

## Program:



Move disk 1 from rod A to rod C Move disk 2 from rod A to rod B Move disk 1 from rod C to rod B Move disk 3 from rod A to rod C Move disk 1 from rod B to rod A Move disk 2 from rod B to rod C Move disk 1 from rod A to rod C

#include <stdio.h>

void towerOfHanoi(int n, char from\_rod, char to\_rod, char aux\_rod) { if (n == 0) {

return;

}

towerOfHanoi(n - 1, from\_rod, aux\_rod, to\_rod);

printf("Move disk %d from rod %c to rod %c\n", n, from\_rod, to\_rod); towerOfHanoi(n - 1, aux\_rod, to\_rod, from\_rod);

}

// Driver code int main() {

int N = 3;

// A, B and C are names of rods towerOfHanoi(N, 'A', 'C', 'B'); return 0;

}

## Output :

**Application:**



1. What is Recursion? Explain in brief.
2. What is Tower of Honoi?

**Date:**

# Practical 5

**AIM: Write a program to implement a queue using arrays and perform Enqueue, Dequeue, and Display operations.**

## Program:

#include <stdio.h>

// Define the maximum size for the queue #define MAX\_SIZE 100

// Define a structure for the queue struct Queue {

int queue[MAX\_SIZE]; int front;

int rear;

};

// Function to initialize the queue

void initializeQueue(struct Queue \*q) { q->front = -1;

q->rear = -1;

}

// Function to check if the queue is empty int isEmpty(struct Queue \*q) {

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

}

// Function to check if the queue is full int isFull(struct Queue \*q) {

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

}

// Function to insert an element into the queue void enqueue(struct Queue \*q, int data) {

if (isFull(q)) { printf("Queue is full\n"); return;

}

if (isEmpty(q)) { q->front = 0;

}

q->rear++;

q->queue[q->rear] = data; printf("Enqueued %d in queue\n", data);

}

// Function to remove an element from the queue int dequeue(struct Queue \*q) {

if (isEmpty(q)) { printf("Queue is empty\n"); return -1;

}

int data = q->queue[q->front];

// If the queue is empty reset the pointers if (q->front == q->rear) {

q->front = -1;

q->rear = -1;



data);

of the queue

} else {

q->front++;

}

printf("Deleted element: %d\n", return data;

}

// Function to display the elements void display(struct Queue \*q) {

if (isEmpty(q)) { printf("Queue is empty\n"); return;

}

for (int i = q->front; i <= q->rear; i++) { printf("%d ", q->queue[i]);

}

printf("\n");

}

int main() {

// Initialize a queue struct Queue q; initializeQueue(&q);

enqueue(&q, 1);

enqueue(&q, 2);

enqueue(&q, 3);

printf("Elements in the queue after enqueue operation: ");

display(&q);



dequeue(&q);

printf("Elements in the queue after dequeue operation: "); display(&q);

return 0;

}

## Output :

Enqueued 1 in queue Enqueued 2 in queue Enqueued 3 in queue

Elements in the queue after enqueue operation: 1 2 3 Deleted element: 1

Elements in the queue after dequeue operation: 2 3

**Application:**

1. What is queue?
2. Write one difference between simple queue and circular queue?

**Date:**

# Practical 6

**AIM: Write a menu driven program to implement following operations on the singly linked list.**

* 1. **Insert a node at the front of the linked list.**
  2. **Insert a node at the end of the linked list.**
  3. **Insert a node such that linked list is in ascending order. (according to info. field)**
  4. **Delete the first node of the linked list.**
  5. **Delete a node with a given value in info. field.**
  6. **Delete a node after specified position.**

## Program:

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

// Define the Node structure struct Node {

int data;

struct Node\* next;

};

// Function to create a new node struct Node\* createNode(int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node)); newNode->data = data;

newNode->next = NULL; return newNode;

}

// Function to insert a new element at the beginning of the singly linked list void insertAtFirst(struct Node\*\* head, int data) {

struct Node\* newNode = createNode(data); newNode->next = \*head;

\*head = newNode;

}

// Function to insert a new element at the end of the singly linked list 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;

}

// Function to insert a new element at a specific position in the singly linked list void insertAtPosition(struct Node\*\* head, int data, int position) {

struct Node\* newNode = createNode(data); if (position == 0) {

insertAtFirst(head,data); return;

}

struct Node\* temp = \*head;

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

}

if (temp == NULL) { printf("Position out of range\n"); free(newNode);

return;

}

newNode->next = temp->next; temp->next = newNode;

}

// Function to delete the first node of the singly linked list void deleteFromFirst(struct Node\*\* head) {

if (\*head == NULL) { printf("List is empty\n"); return;

}

struct Node\* temp = \*head;

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

}

// Function to delete the last node of the singly linked list void deleteFromEnd(struct Node\*\* head) {

if (\*head == NULL) { printf("List is empty\n");

return;

}

struct Node\* temp = \*head; if (temp->next == NULL) {

free(temp);

\*head = NULL; return;

}

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

}

free(temp->next); temp->next = NULL;

}

// Function to delete a node at a specific position in the singly linked list void deleteAtPosition(struct Node\*\* head, int position) {

if (\*head == NULL) { printf("List is empty\n"); return;

}

struct Node\* temp = \*head; if (position == 0) {

deleteFromFirst(head); 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;

}

// Function to print the LinkedList void print(struct Node\* head) {

struct Node\* temp = head; while (temp != NULL) {

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

}

printf("NULL\n");

}

// Driver Code int main() {

struct Node\* head = NULL;

insertAtFirst(&head, 10);

printf("Linked list after inserting the node:10 at the beginning \n"); print(head);

printf("Linked list after inserting the node:20 at the end \n"); insertAtEnd(&head, 20);

print(head);

printf("Linked list after inserting the node:5 at the end \n"); insertAtEnd(&head, 5);

print(head);

printf("Linked list after inserting the node:30 at the end \n"); insertAtEnd(&head, 30);

print(head);

printf("Linked list after inserting the node:15 at position 2 \n"); insertAtPosition(&head, 15, 2);

print(head);

printf("Linked list after deleting the first node: \n"); deleteFromFirst(&head);

print(head);

printf("Linked list after deleting the last node: \n"); deleteFromEnd(&head);

print(head);

printf("Linked list after deleting the node at position 1: \n"); deleteAtPosition(&head, 1);

print(head);

return 0;

}

## Output :



Linked list after inserting the node:10 at the beginning 10 -> NULL

Linked list after inserting the node:20 at the end 10 -> 20 -> NULL

Linked list after inserting the node:5 at the end 10 -> 20 -> 5 -> NULL

Linked list after inserting the node:30 at the end 10 -> 20 -> 5 -> 30 -> NULL

Linked list after inserting the node:15 at position 2 10 -> 20 -> 15 -> 5 -> 30 -> NULL

Linked list after deleting the first node: 20 -> 15 -> 5 -> 30 -> NULL

Linked list after deleting the last node: 20 -> 15 -> 5 -> NULL

Linked list after deleting the node at position 1: 20 -> 5 -> NULL

**Application:**

1. What is Singly Linked List?
2. Write down types of Linked List.

**Date:**

# Practical 7

**AIM: Write a program to implement stack using linked list.**

## Program:

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

// LINKED LIST UTILITY FUNCITON

// Define the structure for a node of the linked list typedef struct Node {

int data;

struct Node\* next;

} node;

// linked list utility function node\* createNode(int data)

{

// allocating memory

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

// if memory allocation is failed if (newNode == NULL)

return NULL;

// putting data in the node newNode->data = data; newNode->next = NULL; return newNode;

}

// fuction to insert data before the head node int insertBeforeHead(node\*\* head, int data)

{

// creating new node

node\* newNode = createNode(data);

// if malloc fail, return error code if (!newNode)

return -1;

// if the linked list is empty if (\*head == NULL) {

\*head = newNode; return 0;

}

newNode->next = \*head;

\*head = newNode; return 0;

}

// deleting head node

int deleteHead(node\*\* head)

{

// no need to check for empty stack as it is already

// being checked in the caller function node\* temp = \*head;

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

return 0;

}

// STACK IMPLEMENTATION STARTS HERE

// Function to check if the stack is empty or not

int isEmpty(node\*\* stack) { return \*stack == NULL; }

// Function to push elements to the stack void push(node\*\* stack, int data)

{

// inserting the data at the beginning of the linked

// list stack

// if the insertion function returns the non - zero

// value, it is the case of stack overflow if (insertBeforeHead(stack, data)) {

printf("Stack Overflow!\n");

}

}

// Function to pop an element from the stack int pop(node\*\* stack)

{

// checking underflow condition if (isEmpty(stack)) {

printf("Stack Underflow\n"); return -1;

}

// deleting the head. deleteHead(stack);

}

// Function to return the topmost element of the stack int peek(node\*\* stack)

{

// check for empty stack if (!isEmpty(stack))

return (\*stack)->data; else

return -1;

}

// Function to print the Stack void printStack(node\*\* stack)

{

node\* temp = \*stack; while (temp != NULL) {

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

}

printf("\n");

}

// driver code int main()

{

// Initialize a new stack top pointer node\* stack = NULL;

// Push elements into the stack push(&stack, 10);

push(&stack, 20);

push(&stack, 30);

push(&stack, 40);

push(&stack, 50);

// Print the stack printf("Stack: "); printStack(&stack);

// Pop elements from the stack pop(&stack);

pop(&stack);

// Print the stack after deletion of elements printf("\nStack: ");

printStack(&stack);

return 0;

}

## Output :



Stack: 50-> 40-> 30-> 20-> 10->

Stack: 30-> 20-> 10->

**Application:**

1. In a stack, which element is removed first?
2. Which end of the linked list is used for push and pop operations in a stack implementation?

**Date:**

# Practical 8

**AIM: Write a program to implement queue using linked list.**

## Program:

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

// Define the structure for a node of the linked list typedef struct Node {

int data;

struct Node\* next;

} node;

// Define the structure for the queue typedef struct Queue {

node\* front; node\* rear;

} queue;

// Function to create a new node node\* createNode(int data)

{

// Allocate memory for a new node

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

// Check if memory allocation was successful if (newNode == NULL)

return NULL;

// Initialize the node's data and next pointer newNode->data = data;

newNode->next = NULL; return newNode;

}

// Function to create a new queue queue\* createQueue()

{

// Allocate memory for a new queue

queue\* newQueue = (queue\*)malloc(sizeof(queue));

// Initialize the front and rear pointers of the queue newQueue->front = newQueue->rear = NULL;

return newQueue;

}

// Function to check if the queue is empty int isEmpty(queue\* q)

{

// Check if the front pointer is NULL return q->front == NULL;

}

// Function to add an element to the queue void enqueue(queue\* q, int data)

{

// Create a new node with the given data node\* newNode = createNode(data);

// Check if memory allocation for the new node was

// successful

if (!newNode) {

printf("Queue Overflow!\n"); return;

}

// If the queue is empty, set the front and rear

// pointers to the new node if (q->rear == NULL) {

q->front = q->rear = newNode; return;

}

// Add the new node at the end of the queue and update

// the rear pointer

q->rear->next = newNode; q->rear = newNode;

}

// Function to remove an element from the queue int dequeue(queue\* q)

{

// Check if the queue is empty if (isEmpty(q)) {

printf("Queue Underflow\n"); return -1;

}

// Store the front node and update the front pointer node\* temp = q->front;

q->front = q->front->next;

// If the queue becomes empty, update the rear pointer

if (q->front == NULL) q->rear = NULL;

// Store the data of the front node and free its memory int data = temp->data;

free(temp); return data;

}

// Function to return the front element of the queue int peek(queue\* q)

{

// Check if the queue is empty if (isEmpty(q))

return -1;

// Return the data of the front node return q->front->data;

}

// Function to print the queue void printQueue(queue\* q)

{

// Traverse the queue and print each element node\* temp = q->front;

while (temp != NULL) { printf("%d -> ", temp->data); temp = temp->next;

}

printf("NULL\n");

}

int main()

{

// Create a new queue queue\* q = createQueue();

// Enqueue elements into the queue enqueue(q, 10);

enqueue(q, 20);

enqueue(q, 30);

enqueue(q, 40);

enqueue(q, 50);

// Print the queue printf("Queue: "); printQueue(q);



// Dequeue elements from the queue dequeue(q);

dequeue(q);

// Print the queue after deletion of elements printf("Queue: ");

printQueue(q);

return 0;

}

## Output :

Queue: 10 -> 20 -> 30 -> 40 -> 50 -> NULL

Queue: 30 -> 40 -> 50 -> NULL

**Application:**

1. Is the queue using a linked list dynamic in size?
2. Can a queue be implemented using a singly linked list?

**Date:**

# Practical 9

**AIM: Write a program to implement following operations on the circular linked list.**

* 1. **Insert a node at the end of the linked list.**
  2. **Insert a node before specified position.**
  3. **Delete a first node of the linked list.**
  4. **Delete a node after specified position.**

## Program:

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

// Node structure struct Node {

int data;

struct Node\* next;

};

// Function to create a new node struct Node\* createNode(int data)

{

struct Node\* newNode

= (struct Node\*)malloc(sizeof(struct Node)); newNode->data = data;

newNode->next = NULL; return newNode;

}

// Function to insert a node at the beginning

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

{

struct Node\* newNode = createNode(data); if (\*head == NULL) {

\*head = newNode; newNode->next = \*head;

}

else {

struct Node\* temp = \*head; while (temp->next != \*head) {

temp = temp->next;

}

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

\*head = newNode;

}

}

// Function to insert a node at the end

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

{

struct Node\* newNode = createNode(data); if (\*head == NULL) {

\*head = newNode; newNode->next = \*head;

}

else {

struct Node\* temp = \*head; while (temp->next != \*head) {

temp = temp->next;

}

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

}

}

// Function to insert a node at a specific position void insertAtPosition(struct Node\*\* head, int data,

int position)

{

struct Node\* newNode = createNode(data); if (\*head == NULL && position == 0) {

\*head = newNode; newNode->next = \*head;

}

else if (position == 0) { insertAtBeginning(head, data);

}

else {

struct Node\* temp = \*head; int i = 0;

while (i < position - 1) { temp = temp->next; i++;

}

newNode->next = temp->next; temp->next = newNode;

}

}

// Function to delete a node from the beginning void deleteFromBeginning(struct Node\*\* head)

{

if (\*head == NULL) { return;

}

else if ((\*head)->next == \*head) { free(\*head);

\*head = NULL;

}

else {

struct Node\* temp = \*head; while (temp->next != \*head) {

temp = temp->next;

}

temp->next = (\*head)->next; struct Node\* toDelete = \*head;

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

}

}

// Function to delete a node from the end void deleteFromEnd(struct Node\*\* head)

{

if (\*head == NULL) { return;

}

else if ((\*head)->next == \*head) { free(\*head);

\*head = NULL;

}

else {

struct Node\* secondLast = \*head;

while (secondLast->next->next != \*head) { secondLast = secondLast->next;

}

struct Node\* last = secondLast->next; secondLast->next = \*head;

free(last);

}

}

// Function to delete a node from a specific position void deleteAtPosition(struct Node\*\* head, int position)

{

if (\*head == NULL) {

return;

}

else if (position == 0) { deleteFromBeginning(head);

}

else {

struct Node\* temp = \*head; int i = 0;

while (i < position - 1) { temp = temp->next; i++;

}

struct Node\* toDelete = temp->next; temp->next = temp->next->next; free(toDelete);

}

}

// Function to traverse and print the circular linked list void traverse(struct Node\* head)

{

if (head == NULL) { return;

}

struct Node\* temp = head; do {

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

} while (temp != head); printf("HEAD\n");

}

// Function to search for a node with a given key int search(struct Node\* head, int key)

{

if (head == NULL) { return 0;

}

struct Node\* temp = head; do {

if (temp->data == key) { return 1; // Key found

}

temp = temp->next;

} while (temp != head); return 0; // Key not found

}

// Driver program int main()

{

struct Node\* head = NULL;

// Insertion insertAtEnd(&head, 10);

insertAtEnd(&head, 20);

insertAtBeginning(&head, 5);

insertAtPosition(&head, 15, 2);

// Traversal

printf("Circular Linked List: "); traverse(head);

// Deletion deleteFromEnd(&head); deleteAtPosition(&head, 1);

// Traversal after deletion

printf("Circular Linked List after deletion: "); traverse(head);

// Searching int key = 10;

if (search(head, key)) {

printf("Element %d is found in the linked list.\n", key);

}

else {

printf(

"Element %d is not found in the linked list.\n", key);

}

return 0;

}

## Output :

Circular Linked List: 5 -> 10 -> 15 -> 20 -> HEAD Circular Linked List after deletion: 5 -> 15 -> HEAD Element 10 is not found in the linked list.



**Application:**

1. What is a circular linked list?
2. How does a circular linked list differ from a singly linked list?

**Date:**

# Practical 10

**AIM: Write a program to implement following operations on the doubly linked list.**

* 1. **Insert a node at the front of the linked list.**
  2. **Insert a node at the end of the linked list.**
  3. **Delete a last node of the linked list.**
  4. **Delete a node before specified position.**

## Program:

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

// defining a node typedef struct Node {

int data;

struct Node\* next; struct Node\* prev;

} Node;

// Function to create a new node with given value as data Node\* createNode(int data)

{

Node\* newNode = (Node\*)malloc(sizeof(Node)); newNode->data = data;

newNode->next = NULL; newNode->prev = NULL; return newNode;

}

// Function to insert a node at the beginning void insertAtBeginning(Node\*\* head, int data)

{

// creating new node

Node\* newNode = createNode(data);

// check if DLL is empty if (\*head == NULL) {

\*head = newNode; return;

}

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

\*head = newNode;

}

// Function to insert a node at the end void insertAtEnd(Node\*\* head, int data)

{

// creating new node

Node\* newNode = createNode(data);

// check if DLL is empty if (\*head == NULL) {

\*head = newNode; return;

}

Node\* temp = \*head;

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

}

temp->next = newNode; newNode->prev = temp;

}

// Function to insert a node at a specified position

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

{

if (position < 1) {

printf("Position should be >= 1.\n"); return;

}

// if we are inserting at head if (position == 1) {

insertAtBeginning(head, data); return;

}

Node\* newNode = createNode(data); Node\* temp = \*head;

for (int i = 1; temp != NULL && i < position - 1; i++) { temp = temp->next;

}

if (temp == NULL) { printf(

"Position greater than the number of nodes.\n"); return;

}

newNode->next = temp->next; newNode->prev = temp;

if (temp->next != NULL) {

temp->next->prev = newNode;

}

temp->next = newNode;

}

// Function to delete a node from the beginning void deleteAtBeginning(Node\*\* head)

{

// checking if the DLL is empty if (\*head == NULL) {

printf("The list is already empty.\n"); return;

}

Node\* temp = \*head;

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

(\*head)->prev = NULL;

}

free(temp);

}

// Function to delete a node from the end void deleteAtEnd(Node\*\* head)

{

// checking if DLL is empty if (\*head == NULL) {

printf("The list is already empty.\n"); return;

}

Node\* temp = \*head;

if (temp->next == NULL) {

\*head = NULL; free(temp); return;

}

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

}

temp->prev->next = NULL;

free(temp);

}

// Function to delete a node from a specified position void deleteAtPosition(Node\*\* head, int position)

{

if (\*head == NULL) {

printf("The list is already empty.\n"); return;

}

Node\* temp = \*head; if (position == 1) {

deleteAtBeginning(head); return;

}

for (int i = 1; temp != NULL && i < position; i++) { temp = temp->next;

}

if (temp == NULL) {

printf("Position is greater than the number of " "nodes.\n");

return;

}

if (temp->next != NULL) {

temp->next->prev = temp->prev;

}

if (temp->prev != NULL) {

temp->prev->next = temp->next;

}

free(temp);

}

// Function to print the list in forward direction void printListForward(Node\* head)

{

Node\* temp = head; printf("Forward List: "); while (temp != NULL) {

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

}

printf("\n");

}

// Function to print the list in reverse direction

void printListReverse(Node\* head)

{

Node\* temp = head; if (temp == NULL) {

printf("The list is empty.\n"); return;

}

// Move to the end of the list while (temp->next != NULL) {

temp = temp->next;

}

// Traverse backwards printf("Reverse List: "); while (temp != NULL) {

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

}

printf("\n");

}

int main()

{

Node\* head = NULL;

// Demonstrating various operations insertAtEnd(&head, 10);

insertAtEnd(&head, 20);

insertAtBeginning(&head, 5);

insertAtPosition(&head, 15, 2); // List: 5 15 10 20

printf("After Insertions:\n"); printListForward(head); printListReverse(head);

deleteAtBeginning(&head); // List: 15 10 20

deleteAtEnd(&head); // List: 15 10

deleteAtPosition(&head, 2); // List: 15

printf("After Deletions:\n"); printListForward(head);

return 0;

}

## Output :



After Insertions: Forward List: 5 15 10 20

Reverse List: 20 10 15 5 After Deletions: Forward List: 15

**Application:**

1. What is doubly linked list?
2. How many pointers does each node in a doubly linked list have?

**Date:**

# Practical 11

**AIM: Write a program which create binary search tree. Also write a function to search an element from binary search tree.**

## Program:

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

struct Node { int key;

struct Node\* left; struct Node\* right;

};

// Constructor to create a new BST node struct Node\* newNode(int item)

{

struct Node\* temp

= (struct Node\*)malloc(sizeof(struct Node)); temp->key = item;

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

}

// function to search a key in a BST

struct Node\* search(struct Node\* root, int key)

{

if (root == NULL || root->key == key) return root;

if (root->key < key)

return search(root->right, key); return search(root->left, key);

}

int main()

{

struct Node\* root = newNode(50); root->left = newNode(30);

root->right = newNode(70); root->left->left = newNode(20);

root->left->right = newNode(40); root->right->left = newNode(60);

root->right->right = newNode(80);



printf(search(root, 19) != NULL ? "Found\n"

: "Not Found\n"); printf(search(root, 80) != NULL ? "Found\n"

: "Not Found\n");

return 0;

}

## Output :

Not Found Found

**Application:**

1. What is Binary Search Tree?
2. What is the time complexity of searching in a BST?

**Date:**

# Practical 12

**AIM: Write a program to implement Bubble Sort.**

## Program:

#include <stdio.h>

void swap(int\* arr, int i, int j) { int temp = arr[i];

arr[i] = arr[j]; arr[j] = temp;

}

void bubbleSort(int arr[], int n) { for (int i = 0; i < n - 1; i++) {

// Last i elements are already in place, so the loop

// will only num n - i - 1 times for (int j = 0; j < n - i - 1; j++) {

if (arr[j] > arr[j + 1])

swap(arr, j, j + 1);

}

}

}

int main() {

int arr[] = { 5, 6, 1, 3 };

int n = sizeof(arr) / sizeof(arr[0]);

// Calling bubble sort on array arr bubbleSort(arr, n);

for (int i = 0; i < n; i++) printf("%d ", arr[i]);

return 0;

}

## Output :

1 3 5 6

**Application:**



1. What is the best case time complexity of Bubble Sort?
2. What is bubble sort?

**Date:**

# Practical 13

**AIM: Write a program to implement Insertion Sort.**

## Program:

#include <math.h> #include <stdio.h>

void insertionSort(int arr[], int N) { for (int i = 1; i < N; i++) {

int key = arr[i]; int j = i - 1;

while (j >= 0 && arr[j] > key) { arr[j + 1] = arr[j];

j = j - 1;

}

// Move the key to its correct position arr[j + 1] = key;

}

}

int main() {

int arr[] = { 12, 11, 13, 5, 6 };

int N = sizeof(arr) / sizeof(arr[0]);

printf("Unsorted array: "); for (int i = 0; i < N; i++) { printf("%d ", arr[i]);

}

printf("\n");

// Calling insertion sort on array arr insertionSort(arr, N);

printf("Sorted array: "); for (int i = 0; i < N; i++) {

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

}

printf("\n"); return 0;

}

## Output :



Unsorted array: 12 11 13 5 6

Sorted array: 5 6 11 12 13

**Application:**

1. What is the worst-case time complexity of Insertion Sort?
2. Is Bubble Sort stable or unstable?

**Date:**

# Practical 14

**AIM: Write a program to create and display records using sequential file organization.**

## Program:

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

int c,i,id;

char name[20]; FILE \*fp;

int n;

int search(FILE \*fp,int id); void display(FILE \*fp);

typedef struct details

{

int id;

char name[20];

}details; details d;

void main()

{

printf("\nHow many records you would like to insert ? : "); scanf("%d",&n);

fp=fopen("one.txt","a"); for(i=0;i<n;i++)

{

printf("\nEnter id and name"); scanf("%d%s",&d.id,d.name);

fwrite(&d,sizeof(d),1,fp);

}

fclose(fp); while(1)

{

printf("\nWhat would you like to do now ? : \n"); printf("\n1.Display \t2.Search \t3.Modify \t4.Delete \t5.Exit"); scanf("%d",&c);

switch(c)

{

case 1:

fp=fopen("one.txt","r+"); display(fp);

fclose(fp); break;

case 2:

fp=fopen("one.txt","r+"); printf("\nEnter ID to search : "); scanf("%d",&id); if(search(fp,id))

{

printf("\nThe record is as follows : "); printf("\n%d\t%s",d.id,d.name);

}

else

printf("\nRecord not found"); fclose(fp);

break; case 3:

fp=fopen("one.txt","r+");

printf("\nEnter ID to modify d record : "); scanf("%d",&id);

if(search(fp,id))

{

printf("\nEnter new name"); scanf("%s",d.name);

fwrite(&d,sizeof(d),1,fp);

}

else

printf("\nSpecified record not found "); fclose(fp);

break;

}

}

}

int search(FILE \*fp,int id)

{

rewind(fp); while(fread(&d,sizeof(d),1,fp))

{

if(id==d.id) return 1;

}

return 0;

}



void display(FILE \*fp)

{

rewind(fp); while(fread(&d,sizeof(d),1,fp))

{

printf("\n%d\t%s",d.id,d.name);

}

}

## Output :

How many records you would like to insert ? : 2

Enter id and name101 sid Segmentation fault

**Application:**

1. What type of access is allowed in sequential file organization?
2. How are records stored in a sequential file?