

**DEPARTMENT OF INFORMATION TECHNOLOGY & COMPUTER
ENGINEERING**



LAB MANUAL

SUBJECT: DATA STRUCTURE LABORATORY

BRANCH: 3RD SEM, IT AND CE & IOT

FACULTY NAME: DR. ANITARANI BRAHMA

Jharsuguda Engineering School, Jharsuguda

Jharsuguda Engineering School, Jharsuguda

LEARNING OUTCOMES:

After completion of the course, the students will be able to:

- ☐ Explain basic terminologies and operations on data structures.
- ☐ Perform asymptotic and worst-case analysis of algorithms.
- ☐ Implement linear data structures.
- ☐ Apply trees and graphs to solve problems.
- ☐ Implement sorting and searching algorithms.

Department of Information Technology

Vision:

To focus on development of skilled and confident personalities of today and tomorrow by using cutting edge technology in the department of Information Technology to accept need based challenges with a sense of social responsibilities.

Mission:

- **M1:** To impart quality education by implementing state-of- the- art teaching-learning methods to enrich the academic competency, credibility and integrity of the students.
- **M2:** To implement the educational program in our department from fundamental engineering to recent technology as per emerging trends in the field of Information Technology.
- **M3:** To facilitate a conducive ambience and infrastructure to develop professional skills and nurture innovation in students.
- **M4:** To inculcate sensitivity towards society, respect for environment and promote high standards of ethics.

PRACTICAL EVALUATION SHEET

BRANCH: _____

NAME: _____

REGD NO.: _____

SUBJECT: _____

EXPERIMENT NO	PERFORMANCE	VIVA	RECORD	TOTAL
1	/3	/2	/2	/7
2	/2	/1	/1	/4
3	/2	/1	/1	/4
4	/2	/1	/1	/4
5	/2	/1	/1	/4
6	/2	/1	/1	/4
7	/2	/1	/1	/4
8	/2	/1	/1	/4
TOTAL				/35

Signature of the faculty

CONTENT

SL NO	EXPERIMENT	PAGE NO	DATE	REMARK
0	Introduction			
	a)Print a message using 'C'			
	b) Program using if-else condition			
	c) Program using loop			
1	Program on Array			
	1.1 Searching			
	1.2 Insert			
2	Program On Stack			
	2.1 Push operation			
	2.2 Push & Pop operation			
	2.3 Infix to postfix conversion			
	2.4 Postfix expression evaluaion			
3	Programs on Queue			
	3.1 Enqueue & Dequeue in simple Queue			
	3.2 Enqueue & Dequeue in circular Queue			
	3.3 Enqueue & Dequeue in Double Ended Queue			
4	Program On Structure			
	4.1 Student Record Management			
5	PROGRAMS ON Pointer			
	5.1 Swapping Two Numbers using Pointers			
	5.2 Pointer with Arrays			
	5.3 String with Pointer			
6	PROGRAMS ON LINKED LIST			
	6.1: CREATE A LINKED LIST OF 3 NODES AND TRAVERSE THE SIMPLE LINKED LIST			
	6.2: insert a node at the beginning of the Linked list			
	6.3: insert a node at the beginning of the Linked list			

	6.4: insert a node at any position of the Linked list			
	6.5: Deletion Operation in Linked list			
	6.6: Creation of doubly Linked list			
	6.7: Insertion Operation in doubly Linked list			
	6.8: Insertion Operation in doubly Linked list			
7	Program on Sorting			
	7.1 Bubble Sort			
	7.2 Selection Sort			
	7.3 Insertion Sort			
	7.4 Merge Sort			
	7.5 QUICK SORT			
8	PROGRAM ON SEARCHING			
	8.1 Searching in BST			
	8.2 Searching in Hash Table			

Experiment1: Programs on Array

PROGRAM 1.1:

//Searching an element in an array

```
#include <stdio.h>

int main()
{
    int arr[50], n, i, search, found = 0;

    // Input array size
    printf("Enter number of elements in array: ");
    scanf("%d", &n);

    // Input array elements
    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    // Input element to search
    printf("Enter element to search: ");
    scanf("%d", &search);

    // Linear Search
```

```
for (i = 0; i < n; i++) {  
    if (arr[i] == search) {  
        printf("Element %d found at position %d.\n", search, i + 1);  
        found = 1;  
        break;  
    }  
}  
  
if (found==0)  
{  
    printf("Element %d not found in the array.\n", search);  
}  
  
return 0;  
}
```

PROGRAM 1.2:

//Inserting an element in an array

```
#include <stdio.h>

int main()
{
    int arr[50], n, i, pos, value;

    // Input size of array
    printf("Enter number of elements in array: ");
    scanf("%d", &n);

    // Input array elements
    for (i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
    }

    // Input position and value to insert
    printf("Enter the position to insert");
    scanf("%d", &pos);
    printf("Enter the value to insert: ");
    scanf("%d", &value);
```

```
// Shift elements to right
for (i = n; i >= pos; i--)
{
    arr[i] = arr[i - 1];
}

// Insert new value
arr[pos - 1] = value;
n++; // increase size

// Display array after insertion
printf("Array after insertion:\n");
for (i = 0; i < n; i++)
{
    printf("%d ", arr[i]);
}

return 0;
}
```

Experiment2: Programs on Stack

PROGRAM 2.1:

/*Stack PUSH operation*/

```
#include <stdio.h>

#define MAXSIZE 5    // Define the maximum size of the stack

int stack[MAXSIZE]; // Array to hold the stack elements

int top = -1;        // Initialize top to -1, meaning stack is empty

// Function to push an element onto the stack

void push(int value)
{
    if (top == MAXSIZE - 1)
    {
        printf("Stack Overflow! Cannot push %d\n", value);
    }
    else
    {
        top++;
        stack[top] = value;
        printf("Pushed %d onto the stack.\n", value);
    }
}
```

```
// Function to display the stack
```

```
void display()
```

```
{
```

```
    if (top == -1)
```

```
{
```

```
    printf("Stack is empty.\n");
```

```
}
```

```
else
```

```
{
```

```
    printf("Stack elements: ");
```

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

```
{
```

```
        printf("%d ", stack[i]);
```

```
}
```

```
    printf("\n");
```

```
}
```

```
}
```

```
int main()
```

```
{
```

```
    push(10);
```

```
    push(20);  
    push(30);  
    display();  
    return 0;  
}
```

PROGRAM 2.2:

/*Stack PUSH & POP operation*/

```
#include <stdio.h>  
  
#define MAXSIZE SIZE 5 // Maximum size of the stack  
  
int stack[MAXSIZE]; // Array to store stack elements  
  
int top = -1; // Initialize top to -1 (empty stack)  
  
// Function to pop an element from the stack  
  
void pop()  
{  
    if (top == -1)  
    {  
        printf("Stack Underflow! Cannot pop element.\n");  
    }  
}
```

```
    }  
    else  
    {  
        int poppedValue = stack[top];  
        top--;  
        printf("Popped %d from the stack.\n", poppedValue);  
    }  
}
```

// Function to push an element (for testing pop)

```
void push(int value)  
{  
    if (top == MAX SIZE - 1)  
    {  
        printf("Stack Overflow! Cannot push %d\n", value);  
    }  
    else  
    {  
        top++;  
        stack[top] = value;  
        printf("Pushed %d onto the stack.\n", value);  
    }  
}
```

```
}
```

```
// Function to display the current stack
```

```
void display()
```

```
{
```

```
    if (top == -1)
```

```
    {
```

```
        printf("Stack is empty.\n");
```

```
    }
```

```
Else
```

```
{
```

```
    printf("Stack elements: ");
```

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

```
        printf("%d ", stack[i]);
```

```
    }
```

```
    printf("\n");
```

```
}
```

```
}
```

```
int main() {
```

```
    // Example operations
```

```
    push(10);
```

```
    push(20);
```

```
    push(30);  
    display();  
    pop();  
    display();  
    pop();  
    display();  
    return 0;  
}
```

PROGRAM 2.3:

/*CONVERSION OF INFIX TO POSTFIX EXPRESSION*/

```
#include <stdio.h>  
  
#include <ctype.h> // for isalnum()  
  
#include <string.h> // for strlen()  
  
#define MAX 100  
  
// Stack implementation  
char stack[MAX];  
  
int top = -1;  
  
// Function to push an element into the stack  
void push(char c)  
{  
    if (top == MAX - 1)  
{
```

```

        printf("Stack Overflow\n");
    }
else
{
    stack[++top] = c;
}
}

// Function to pop an element from the stack
char pop() {
    if (top == -1) {
        return -1; // Stack underflow
    } else {
        return stack[top--];
    }
}

// Function to return priority of operators
int precedence(char c) {
    if (c == '^')
        return 3;
    else if (c == '*' || c == '/')

```

```
        return 2;
    else if (c == '+' || c == '-')
        return 1;
    else
        return 0;
}
```

// Function to convert Infix to Postfix

```
void infixToPostfix(char infix[])
{
    char postfix[MAX];
    int i, k = 0;
    char symbol;
    for (i = 0; i < strlen(infix); i++)
    {
        symbol = infix[i];
        // If operand, add to postfix
        if (isalnum(symbol))
        {
            postfix[k++] = symbol;
        }
        // If '(', push to stack
```

```
else if (symbol == '(')
{
    push(symbol);
}

// If ')', pop until '('
else if (symbol == ')')
{
    while (top != -1 && stack[top] != '(')
    {
        postfix[k++] = pop();
    }
    pop();
    // Remove '('
}

// If operator
else
{
    while (top != -1 && precedence(stack[top]) >= precedence(symbol))
    {
        postfix[k++] = pop();
    }
    push(symbol);
}
```

```

    }
}

// Pop all remaining operators
while (top != -1) {
    postfix[k++] = pop();
}

postfix[k] = '\0'; // Null terminate string
printf("Postfix Expression: %s\n", postfix);
}

// Main function
int main() {
    char infix[MAX];

    printf("Enter an Infix Expression: ");

    gets(infix); // (For simplicity; unsafe, but okay for learning)

    infixToPostfix(infix);

    return 0;
}

```

NOTE: The `isalnum()` function returns a non-zero value (equivalent to boolean true) if a character is alphanumeric, meaning an alphabet letter (a-z) or a number (0-9).

Example of characters that are not alphanumeric: (space)! #%&?

PROGRAM 2.4:

/*EVALUATION OF POSTFIX EXPRESSION*/

```
#include <stdio.h>
```

```
#include <ctype.h> // for isdigit()
```

```
#define MAX 100
```

```
int stack[MAX];
```

```
int top = -1;
```

```
// Push into stack
```

```
void push(int x)
```

```
{
```

```
    stack[++top] = x;
```

```
}
```

```
// Pop from stack
```

```
int pop()
```

```
{
```

```
    return stack[top--];
```

```
}
```

```
// Function to evaluate Postfix expression
```

```
int evaluatePostfix(char exp[])
```

```
{
```

```
int i;

char c;

int val;

for (i = 0; exp[i] != '\0'; i++)
{
    c = exp[i];

    // If operand (digit), push to stack
    if (isdigit(c))
    {
        push(c - '0'); // convert char to int
    }

    // If operator, pop two operands and apply operator
    else
    {
        int val2 = pop();
        int val1 = pop();
        switch (c)
        {
            case '+': val = val1 + val2; break;
            case '-': val = val1 - val2; break;
            case '*': val = val1 * val2; break;
            case '/': val = val1 / val2; break;
```

```
    }
    push(val);
}
}
return pop(); // Final result
}
// Main function
int main()
{
    char exp[MAX];
    printf("Enter a Postfix Expression (single digit operands): ");
    scanf("%s", exp);
    int result = evaluatePostfix(exp);
    printf("Result of Postfix Evaluation: %d\n", result);
    return 0;
}
```

Experiment 3: Programs on Queue

Program 3.1: Enqueue (Insertion) and Dequeue (Deletion) operation in Queue

```
#include <stdio.h>

#define MAX 5 // maximum size of queue

int queue[MAX];

int front = -1, rear = -1;

// Function to insert (Enqueue) an element
void enqueue(int value) {
    if (rear == MAX - 1) {
        printf("Queue is Full (Overflow)\n");
    } else {
        if (front == -1) {
            front = 0; // first element
```

```

    }
    rear++;
    queue[rear] = value;
    printf("Inserted %d into the queue.\n", value);
}
}
// Function to delete (Dequeue) an element
void dequeue() {
    if (front == -1 || front > rear) {
        printf("Queue is Empty (Underflow)\n");
    } else {
        printf("Deleted element: %d\n", queue[front]);
        front++;
        if (front > rear) {
            // Reset queue when all elements are deleted
            front = rear = -1;
        }
    }
}
}

```

```

// Function to display queue
void display() {

```

```
if (front == -1) {  
    printf("Queue is Empty\n");  
} else {  
    printf("Queue elements: ");  
    for (int i = front; i <= rear; i++) {  
        printf("%d ", queue[i]);  
    }  
    printf("\n");  
}  
}
```

// Main function

```
int main() {  
    int choice, value;  
    do {  
        printf("\n--- Queue Menu ---\n");  
        printf("1. Enqueue (Insert)\n");  
        printf("2. Dequeue (Delete)\n");  
        printf("3. Display Queue\n");  
        printf("4. Exit\n");  
        printf("Enter your choice: ");  
        scanf("%d", &choice);
```

```
switch(choice) {  
    case 1:  
        printf("Enter value to insert: ");  
        scanf("%d", &value);  
        enqueue(value);  
        break;  
    case 2:  
        dequeue();  
        break;  
    case 3:  
        display();  
        break;  
    case 4:  
        printf("Exiting program...\n");  
        break;  
    default:  
        printf("Invalid choice! Try again.\n");  
}  
} while(choice != 4);  
  
return 0;
```

```
}
```

Program 3.2: Enqueue (Insertion) and Dequeue (Deletion) operation in Circular Queue

```
#include <stdio.h>

#define MAX 5 // Maximum size of Circular Queue

int queue[MAX];

int front = -1, rear = -1;

// Function to insert (Enqueue) an element
void enqueue(int value) {
    if ((front == 0 && rear == MAX - 1) || (front == rear + 1)) {
        printf("Queue is Full (Overflow)\n");
    } else {
        if (front == -1) {
            front = rear = 0; // first element
        } else {
```

```
        rear = (rear + 1) % MAX;
    }
    queue[rear] = value;
    printf("Inserted %d into Circular Queue.\n", value);
}
}
```

// Function to delete (Dequeue) an element

```
void dequeue() {
    if (front == -1) {
        printf("Queue is Empty (Underflow)\n");
    } else {
        printf("Deleted element: %d\n", queue[front]);
        if (front == rear) {
            front = rear = -1; // Queue becomes empty
        } else {
            front = (front + 1) % MAX;
        }
    }
}
```

// Function to display Circular Queue

```
void display() {  
    if (front == -1) {  
        printf("Queue is Empty\n");  
    } else {  
        printf("Circular Queue elements: ");  
        int i = front;  
        while (1) {  
            printf("%d ", queue[i]);  
            if (i == rear) break;  
            i = (i + 1) % MAX;  
        }  
        printf("\n");  
    }  
}
```

// Main function

```
int main() {  
    int choice, value;  
    do {  
        printf("\n--- Circular Queue Menu ---\n");  
        printf("1. Enqueue (Insert)\n");  
        printf("2. Dequeue (Delete)\n");
```

```
printf("3. Display Queue\n");  
printf("4. Exit\n");  
printf("Enter your choice: ");  
scanf("%d", &choice);  
  
switch(choice) {  
    case 1:  
        printf("Enter value to insert: ");  
        scanf("%d", &value);  
        enqueue(value);  
        break;  
    case 2:  
        dequeue();  
        break;  
    case 3:  
        display();  
        break;  
    case 4:  
        printf("Exiting program...\n");  
        break;  
    default:  
        printf("Invalid choice! Try again.\n");
```

```

    }
} while(choice != 4);

return 0;
}

```

Program 3.3: Enqueue (Insertion) and Dequeue (Deletion) operation in Double Ended Queue

```

#include <stdio.h>

#define MAX 5 // maximum size of deque

int deque[MAX];

int front = -1, rear = -1;

// Function to insert element at front
void insertFront(int value) {
    if ((front == 0 && rear == MAX-1) || (front == rear+1)) {
        printf("Deque is Full (Overflow)\n");
    } else {
        if (front == -1) { // first element
            front = rear = 0;
        } else if (front == 0) {

```

```
        front = MAX - 1;
    } else {
        front = front - 1;
    }
    deque[front] = value;
    printf("Inserted %d at front\n", value);
}
}
```

// Function to insert element at rear

```
void insertRear(int value) {
    if ((front == 0 && rear == MAX-1) || (front == rear+1)) {
        printf("Deque is Full (Overflow)\n");
    } else {
        if (front == -1) { // first element
            front = rear = 0;
        } else if (rear == MAX-1) {
            rear = 0;
        } else {
            rear = rear + 1;
        }
        deque[rear] = value;
    }
}
```

```
        printf("Inserted %d at rear\n", value);  
    }  
}
```

// Function to delete element from front

```
void deleteFront() {  
    if (front == -1) {  
        printf("Deque is Empty (Underflow)\n");  
    } else {  
        printf("Deleted %d from front\n", deque[front]);  
        if (front == rear) { // only one element  
            front = rear = -1;  
        } else if (front == MAX-1) {  
            front = 0;  
        } else {  
            front = front + 1;  
        }  
    }  
}
```

// Function to delete element from rear

```
void deleteRear() {
```

```
if (front == -1) {  
    printf("Deque is Empty (Underflow)\n");  
} else {  
    printf("Deleted %d from rear\n", deque[rear]);  
    if (front == rear) { // only one element  
        front = rear = -1;  
    } else if (rear == 0) {  
        rear = MAX - 1;  
    } else {  
        rear = rear - 1;  
    }  
}  
}
```

```
// Function to display deque  
void display() {  
    if (front == -1) {  
        printf("Deque is Empty\n");  
    } else {  
        printf("Deque elements: ");  
        int i = front;  
        while (1) {
```

```
        printf("%d ", deque[i]);  
        if (i == rear) break;  
        i = (i + 1) % MAX;  
    }  
    printf("\n");  
}  
}  
  
// Main function  
int main() {  
    int choice, value;  
    do {  
        printf("\n--- Deque Menu ---\n");  
        printf("1. Insert at Front\n");  
        printf("2. Insert at Rear\n");  
        printf("3. Delete from Front\n");  
        printf("4. Delete from Rear\n");  
        printf("5. Display Deque\n");  
        printf("6. Exit\n");  
        printf("Enter your choice: ");  
        scanf("%d", &choice);
```

```
switch(choice) {  
    case 1:  
        printf("Enter value to insert at front: ");  
        scanf("%d", &value);  
        insertFront(value);  
        break;  
    case 2:  
        printf("Enter value to insert at rear: ");  
        scanf("%d", &value);  
        insertRear(value);  
        break;  
    case 3:  
        deleteFront();  
        break;  
    case 4:  
        deleteRear();  
        break;  
    case 5:  
        display();  
        break;  
    case 6:  
        printf("Exiting program...\n");
```

```
        break;
    default:
        printf("Invalid choice! Try again.\n");
    }
} while(choice != 6);

return 0;
}
```

EXPERIMENT NO. 4: PROGRAMS ON STRUCTURE

Experiment 4.1 Student Record Management

Application: Used in schools/colleges to store and process student information.

```
#include <stdio.h>
```

```
struct Student {
```

```
    int rollNo;
```

```
    char name[30];
```

```
    float marks;
```

```
};
```

```
int main()
```

```
{
```

```
    struct Student s1;
```

```
printf("Enter Roll No: ");
scanf("%d", &s1.rollNo);
printf("Enter Name: ");
scanf("%s", s1.name);
printf("Enter Marks: ");
scanf("%f", &s1.marks);
printf("\n--- Student Details ---\n");
printf("Roll No: %d\n", s1.rollNo);
printf("Name : %s\n", s1.name);
printf("Marks : %.2f\n", s1.marks);
return 0;
}
```

Employee Payroll System

Application: Used in companies to maintain employee salary records.

```
#include <stdio.h>

struct Employee
{
    int id;
    char name[30];
    float salary;
```

```
};
```

```
int main()
```

```
{
```

```
    struct Employee e1;
```

```
    printf("Enter Employee ID: ");
```

```
    scanf("%d", &e1.id);
```

```
    printf("Enter Employee Name: ");
```

```
    scanf("%s", e1.name);
```

```
    printf("Enter Salary: ");
```

```
    scanf("%f", &e1.salary);
```

```
    printf("\n--- Employee Record ---\n");
```

```
    printf("ID    : %d\n", e1.id);
```

```
    printf("Name  : %s\n", e1.name);
```

```
    printf("Salary : %.2f\n", e1.salary);
```

```
    return 0;
```

```
}
```

Library Book Management

Application: Used in libraries to store details of books.

```
#include <stdio.h>

struct Book
{
    int bookID;
    char title[50];
    char author[30];
};

int main()
{
    struct Book b1;
    printf("Enter Book ID: ");
    scanf("%d", &b1.bookID);
    printf("Enter Book Title: ");
    scanf("%s", b1.title);
    printf("Enter Author: ");
    scanf("%s", b1.author);
    printf("\n--- Book Details ---\n");
    printf("Book ID : %d\n", b1.bookID);
    printf("Title  : %s\n", b1.title);
    printf("Author : %s\n", b1.author);
    return 0;
```

```
}
```

Bank Account Management

Application: Used in banks to maintain account information.

```
#include <stdio.h>
```

```
struct BankAccount
```

```
{
```

```
    int accNo;
```

```
    char name[30];
```

```
    float balance;
```

```
};
```

```
int main()
```

```
{
```

```
    struct BankAccount acc1;
```

```
    printf("Enter Account Number: ");
```

```
    scanf("%d", &acc1.accNo);
```

```
printf("Enter Account Holder Name: ");
scanf("%s", acc1.name);
printf("Enter Balance: ");
scanf("%f", &acc1.balance);
printf("\n--- Account Details ---\n");
printf("Account No : %d\n", acc1.accNo);
printf("Name      : %s\n", acc1.name);
printf("Balance   : %.2f\n", acc1.balance);
return 0;
}
```

EXPERIMENT NO. 5: PROGRAMS ON Pointer

5.1 Swapping Two Numbers using Pointers

```
#include <stdio.h>

void swap(int *x, int *y)
{
    int temp = *x;
    *x = *y;
    *y = temp;
}

int main()
```

```
{  
    int a = 5, b = 10;  
    printf("Before Swap: a = %d, b = %d\n", a, b);  
    swap(&a, &b);  
    printf("After Swap : a = %d, b = %d\n", a, b);  
    return 0;  
}
```

5.2 Pointer with Arrays

```
#include <stdio.h>  
  
int main()  
{  
    int arr[5] = {10, 20, 30, 40, 50};  
    int *p = arr; // pointer points to first element  
    printf("Array elements using pointer:\n");  
    for (int i = 0; i < 5; i++)  
    {  
        printf("%d ", *(p + i));  
    }  
    return 0;  
}
```

5.3 String with Pointer

```
#include <stdio.h>

int main()
{
    char *str = "Hello, Pointers!";
    printf("String: %s\n", str);
    while (*str != '\0')
    {
        printf("%c ", *str);
        str++;
    }

    return 0;
}
```

EXPERIMENT NO. 6: PROGRAMS ON LINKED LIST

PROGRAM 6.1: CREATE A LINKED LIST OF 3 NODES AND TRAVERSE THE SIMPLE LINKED LIST

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
// Define structure for a node
```

```
struct Node {
```

```
    int data;
```

```
    struct Node* next;
```

```
};
```

```
int main() {
```

```
    // Declare three node pointers
```

```
    struct Node *head, *second, *third;
```

```
    // Allocate memory for three nodes in the linked list
```

```
head = (struct Node*)malloc(sizeof(struct Node));
second = (struct Node*)malloc(sizeof(struct Node));
third = (struct Node*)malloc(sizeof(struct Node));

// Assign data and link nodes

head->data = 10;    // assign data to first node
head->next = second; // link first node with second

second->data = 20;  // assign data to second node
second->next = third; // link second node with third
third->data = 30;    // assign data to third node
third->next = NULL;  // last node should point to NULL

// Traverse the linked list

printf("Linked List elements:\n");
struct Node* temp = head;
while (temp != NULL) {
    printf("%d -> ", temp->data);
    temp = temp->next;
}
printf("NULL\n");

// Free allocated memory

free(head);
free(second);
```

```
    free(third);

    return 0;

}
```

Explanation:

1. **Structure Definition:**
struct Node contains an integer data and a pointer next to the next node.
2. **Memory Allocation:**
malloc() dynamically allocates memory for each node.
3. **Linking Nodes:**
Each node's next pointer connects to the next node in the sequence.
4. **Traversal:**
A temporary pointer temp is used to traverse and print the list.

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

Program 6.2: insert a node at the beginning of the Linked list

Define structure for a node

```
struct Node {

    int data;

    struct Node* next;

};


// Function to print the linked list

void display(struct Node* head) {

    struct Node* temp = head;

    printf("Linked List: ");
```

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

// Function to insert a new node at the beginning

```
struct Node* insertAtBeginning(struct Node* head, int newData) {  
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));  
    newNode->data = newData;  
    newNode->next = head;  
    return newNode; // new node becomes the new head  
}
```

```
int main() {
```

```
    struct Node *head, *second, *third;
```

// Allocate memory for three nodes

```
head = (struct Node*)malloc(sizeof(struct Node));
```

```
second = (struct Node*)malloc(sizeof(struct Node));
```

```
third = (struct Node*)malloc(sizeof(struct Node));
```

```
// Assign data and link nodes
head->data = 10;
head->next = second;

second->data = 20;
second->next = third;

third->data = 30;
third->next = NULL;

printf("Original ");
display(head);

// Insert a new node at the beginning
head = insertAtBeginning(head, 5);

printf("After Insertion ");
display(head);

return 0;
}
```

Program 6.3: insert a node at the beginning of the Linked list

```
#include <stdio.h>

#include <stdlib.h>

// Define structure for a node
struct Node {
    int data;
    struct Node* next;
};

// Function to print the linked list
void display(struct Node* head) {
    struct Node* temp = head;
    printf("Linked List: ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
```

```
// Function to insert a new node at the end

void insertAtEnd(struct Node* head, int newData) {

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

    newNode->data = newData;

    newNode->next = NULL;


    struct Node* temp = head;

    while (temp->next != NULL) { // Traverse to the last node

        temp = temp->next;

    }

    temp->next = newNode; // Link last node to the new node

}


int main() {

    struct Node *head, *second, *third;


    // Allocate memory for three nodes

    head = (struct Node*)malloc(sizeof(struct Node));

    second = (struct Node*)malloc(sizeof(struct Node));

    third = (struct Node*)malloc(sizeof(struct Node));


    // Assign data and link nodes
```

```
head->data = 10;
head->next = second;

second->data = 20;
second->next = third;

third->data = 30;
third->next = NULL;

printf("Original ");
display(head);

// Insert new node at the end
insertAtEnd(head, 40);

printf("After Insertion ");
display(head);

return 0;
}
```

Program 6.4: insert a node at any position of the Linked list

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
// Define structure for a node
```

```
struct Node {
```

```
    int data;
```

```
    struct Node* next;
```

```
};
```

```
// Function to display the linked list
```

```
void display(struct Node* head) {
```

```
    struct Node* temp = head;
```

```
    printf("Linked List: ");
```

```
    while (temp != NULL) {
```

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

```
        temp = temp->next;
```

```
    }
```

```
    printf("NULL\n");
```

```
}
```

```
// Function to insert a new node at any position
```

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

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

```
newNode->data = data;
```

```
// Case 1: Insert at beginning
```

```
if (position == 1) {
```

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

```
    *head = newNode;
```

```
    return;
```

```
}
```

```
struct Node* temp = *head;
```

```
// Traverse to (position-1)th node
```

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

```
    temp = temp->next;
```

```
}
```

```
// If position is out of range
```

```
if (temp == NULL) {
```

```
    printf("Position out of range!\n");
```

```
    free(newNode);
```

```
    return;
```

```
}
```

```
// Insert the new node after temp  
newNode->next = temp->next;  
temp->next = newNode;  
}
```

```
int main() {  
    struct Node *head, *second, *third;  
  
    // Allocate memory for three nodes  
    head = (struct Node*)malloc(sizeof(struct Node));  
    second = (struct Node*)malloc(sizeof(struct Node));  
    third = (struct Node*)malloc(sizeof(struct Node));  
  
    // Assign data and link nodes  
    head->data = 10;  
    head->next = second;  
  
    second->data = 20;  
    second->next = third;  
  
    third->data = 30;  
    third->next = NULL;
```

```
printf("Original ");  
display(head);  
  
// Insert new nodes at different positions  
insertAtPosition(&head, 5, 1); // At beginning  
insertAtPosition(&head, 15, 3); // In the middle  
insertAtPosition(&head, 40, 6); // At the end  
  
printf("After Insertion ");  
display(head);  
  
return 0;  
}
```

Program 6.5: Deletion Operation in Linked list

```
#include <stdio.h>  
  
#include <stdlib.h>  
  
// Define node structure  
struct Node  
{
```

```
int data;

struct Node *next;

};

struct Node *head = NULL;

// Function to create a linked list

void createList(int n)

{

    struct Node *newNode, *temp;

    int data, i;

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

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

        printf("Enter data for node %d: ", i + 1);

        scanf("%d", &data);

        newNode->data = data;

        newNode->next = NULL;

        if (head == NULL)

            head = newNode;

        else {

            temp = head;
```

```
        while (temp->next != NULL)

            temp = temp->next;

            temp->next = newNode;

        }

    }

}
```

// Function to display the linked list

```
void display() {

    struct Node *temp = head;

    if (head == NULL) {

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

        return;

    }

    printf("Linked List: ");

    while (temp != NULL) {

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

        temp = temp->next;

    }

    printf("NULL\n");

}
```

// Function to delete a node at given position

```
void deleteNode(int pos) {  
    struct Node *temp = head, *prev;  
    int i;  
  
    if (head == NULL) {  
        printf("List is empty!\n");  
        return;  
    }  
  
    if (pos == 1) { // delete first node  
        head = head->next;  
        free(temp);  
        printf("Node deleted from position 1\n");  
        return;  
    }  
  
    for (i = 1; i < pos && temp != NULL; i++) {  
        prev = temp;  
        temp = temp->next;  
    }
```

```
if (temp == NULL) {  
    printf("Position not found!\n");  
    return;  
}
```

```
prev->next = temp->next;  
free(temp);  
printf("Node deleted from position %d\n", pos);  
}
```

```
// Main function
```

```
int main() {  
    int n, pos;  
  
    printf("Enter number of nodes: ");  
    scanf("%d", &n);  
  
    createList(n);  
    display();  
  
    printf("Enter position to delete: ");  
    scanf("%d", &pos);
```

```
    deleteNode(pos);  
    display();  
  
    return 0;  
}
```

Program 6.6: Creation of doubly Linked list

```
#include <stdio.h>  
  
#include <stdlib.h>  
  
// Structure of a doubly linked list node  
struct Node {  
    int data;  
    struct Node *prev;  
    struct Node *next;  
};  
  
struct Node *head = NULL;  
  
// Function to create a doubly linked list  
void createList(int n) {  
    struct Node *newNode, *temp;
```

```
int data, i;
```

```
if (n <= 0) {
```

```
    printf("Number of nodes should be greater than 0.\n");
```

```
    return;
```

```
}
```

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

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

```
    printf("Enter data for node %d: ", i);
```

```
    scanf("%d", &data);
```

```
    newNode->data = data;
```

```
    newNode->prev = NULL;
```

```
    newNode->next = NULL;
```

```
if (head == NULL) {
```

```
    head = newNode;
```

```
} else {
```

```
    temp = head;
```

```
    while (temp->next != NULL)
```

```
        temp = temp->next;
```

```

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

// Function to display the doubly linked list
void displayList() {
    struct Node *temp = head;

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

    printf("Doubly Linked List: ");
    while (temp != NULL) {
        printf("%d <-> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

```

```
// Main function
int main() {
    int n;

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

    createList(n);
    displayList();

    return 0;
}
```

Program 6.7: Insertion Operation in doubly Linked list

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

// Structure of a doubly linked list node
struct Node {
    int data;
    struct Node *prev;
```

```
    struct Node *next;
};

struct Node *head = NULL;

// Function to create a new node
struct Node* createNode(int data) {
    struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

// Function to insert node at beginning
void insertAtBeginning(int data) {
    struct Node *newNode = createNode(data);

    if (head == NULL)
        head = newNode;
    else {
        newNode->next = head;
        head->prev = newNode;
        head = newNode;
    }
}
```

```
    printf("Node inserted at beginning.\n");
}

// Function to insert node at end
void insertAtEnd(int data) {
    struct Node *newNode = createNode(data);
    struct Node *temp;
    if (head == NULL) {
        head = newNode;
        printf("Node inserted as first node.\n");
        return;
    }
    temp = head;
    while (temp->next != NULL)
        temp = temp->next;

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

    printf("Node inserted at end.\n");
}
```

// Function to insert node at any position

```
void insertAtPosition(int data, int pos) {  
    struct Node *newNode = createNode(data);  
    struct Node *temp = head;  
    int i;  
  
    if (pos == 1) {  
        insertAtBeginning(data);  
        return;  
    }  
  
    for (i = 1; i < pos - 1 && temp != NULL; i++)  
        temp = temp->next;  
  
    if (temp == NULL) {  
        printf("Position not found!\n");  
        free(newNode);  
        return;  
    }  
  
    newNode->next = temp->next;
```

```
newNode->prev = temp;
```

```
if (temp->next != NULL)
```

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

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

```
printf("Node inserted at position %d.\n", pos);
```

```
}
```

```
// Function to display the doubly linked list
```

```
void displayList() {
```

```
    struct Node *temp = head;
```

```
    if (head == NULL) {
```

```
        printf("List is empty!\n");
```

```
        return;
```

```
    }
```

```
    printf("Doubly Linked List: ");
```

```
    while (temp != NULL) {
```

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

```
        temp = temp->next;
    }
    printf("NULL\n");
}
```

// Main function

```
int main() {
    int choice, data, pos;

    while (1) {
        printf("\n--- Doubly Linked List Insertion Menu ---\n");
        printf("1. Insert at Beginning\n");
        printf("2. Insert at End\n");
        printf("3. Insert at Any Position\n");
        printf("4. Display List\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data: ");
```

```
scanf("%d", &data);  
insertAtBeginning(data);  
break;
```

case 2:

```
printf("Enter data: ");  
scanf("%d", &data);  
insertAtEnd(data);  
break;
```

case 3:

```
printf("Enter data: ");  
scanf("%d", &data);  
printf("Enter position: ");  
scanf("%d", &pos);  
insertAtPosition(data, pos);  
break;
```

case 4:

```
displayList();  
break;
```

case 5:

```
exit(0);
```

default:

```
printf("Invalid choice!\n");
```

```
    }  
}  
return 0;  
}
```

Program 6.8: Insertion Operation in doubly Linked list

```
#include <stdio.h>  
  
#include <stdlib.h>  
  
// Structure of a doubly linked list node  
struct Node {  
    int data;  
    struct Node *prev;  
    struct Node *next;  
};  
  
struct Node *head = NULL;  
  
// Function to create a new node  
struct Node* createNode(int data) {
```

```
    struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));  
    newNode->data = data;  
    newNode->prev = NULL;  
    newNode->next = NULL;  
    return newNode;  
}
```

// Function to insert node at end (for initial list creation)

```
void insertAtEnd(int data) {  
    struct Node *newNode = createNode(data);  
    struct Node *temp;  
  
    if (head == NULL) {  
        head = newNode;  
        return;  
    }
```

```
    temp = head;  
    while (temp->next != NULL)  
        temp = temp->next;  
  
    temp->next = newNode;
```

```
    newNode->prev = temp;
}
```

// Function to display the list

```
void displayList() {
    struct Node *temp = head;
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }
```

```
    printf("Doubly Linked List: ");
    while (temp != NULL) {
        printf("%d <-> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
```

// Function to delete node from beginning

```
void deleteAtBeginning() {
    struct Node *temp = head;
```

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

```
head = head->next;  
  
if (head != NULL)  
    head->prev = NULL;
```

```
printf("Node deleted from beginning (data = %d)\n", temp->data);  
free(temp);  
}
```

// Function to delete node from end

```
void deleteAtEnd() {  
    struct Node *temp = head;  
  
    if (head == NULL) {  
        printf("List is empty!\n");  
        return;  
    }
```

```
// If only one node
```

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

```
    printf("Node deleted from end (data = %d)\n", head->data);
```

```
    free(head);
```

```
    head = NULL;
```

```
    return;
```

```
}
```

```
while (temp->next != NULL)
```

```
    temp = temp->next;
```

```
temp->prev->next = NULL;
```

```
printf("Node deleted from end (data = %d)\n", temp->data);
```

```
free(temp);
```

```
}
```

```
// Function to delete node from any position
```

```
void deleteAtPosition(int pos) {
```

```
    struct Node *temp = head;
```

```
    int i;
```

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

// Deletion from first position

```
if (pos == 1) {  
    deleteAtBeginning();  
    return;  
}
```

```
for (i = 1; i < pos && temp != NULL; i++)  
    temp = temp->next;
```

```
if (temp == NULL) {  
    printf("Position not found!\n");  
    return;  
}
```

```
if (temp->next != NULL)  
    temp->next->prev = temp->prev;
```

```
if (temp->prev != NULL)
    temp->prev->next = temp->next;

printf("Node deleted from position %d (data = %d)\n", pos, temp->data);
free(temp);
}
```

```
// Main function
```

```
int main() {
    int choice, data, pos, n, i;

    printf("Enter number of nodes to create: ");
    scanf("%d", &n);

    for (i = 1; i <= n; i++) {
        printf("Enter data for node %d: ", i);
        scanf("%d", &data);
        insertAtEnd(data);
    }

    while (1) {
        printf("\n--- Doubly Linked List Deletion Menu ---\n");
```

```
printf("1. Delete from Beginning\n");  
printf("2. Delete from End\n");  
printf("3. Delete from Any Position\n");  
printf("4. Display List\n");  
printf("5. Exit\n");  
printf("Enter your choice: ");  
scanf("%d", &choice);
```

```
switch (choice) {  
    case 1:  
        deleteAtBeginning();  
        break;  
    case 2:  
        deleteAtEnd();  
        break;  
    case 3:  
        printf("Enter position to delete: ");  
        scanf("%d", &pos);  
        deleteAtPosition(pos);  
        break;  
    case 4:  
        displayList();
```

```
        break;
    case 5:
        exit(0);
    default:
        printf("Invalid choice!\n");
    }
}

return 0;
}
```

EXPERIMENT 7: SORTING

7.1 BUBBLE SORT

```
#include <stdio.h>

int main() {

    int arr[100], n, i, j, temp;

    // Input size of array
    printf("Enter number of elements: ");
    scanf("%d", &n);

    // Input array elements
    printf("Enter %d elements:\n", n);
    for(i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
```

```
}
```

```
// Bubble Sort logic
```

```
for(i = 0; i < n-1; i++) {
```

```
    for(j = 0; j < n-i-1; j++) {
```

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

```
            // Swap the elements
```

```
            temp = arr[j];
```

```
            arr[j] = arr[j+1];
```

```
            arr[j+1] = temp;
```

```
        }
```

```
    }
```

```
}
```

```
// Display sorted array
```

```
printf("\nSorted array in ascending order:\n");
```

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

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

```
}
```

```
return 0;
```

```
}
```

7.2 SELECTION SORT

```
#include <stdio.h>

int main() {
    int arr[100], n, i, j, min, temp;

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

    printf("Enter %d elements:\n", n);
    for(i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    // Selection Sort logic
    for(i = 0; i < n-1; i++) {
        min = i;
        for(j = i+1; j < n; j++) {
            if(arr[j] < arr[min])
                min = j;
        }
    }
}
```

```

        // Swap the smallest element with the first unsorted element
        temp = arr[i];
        arr[i] = arr[min];
        arr[min] = temp;
    }

    printf("\nSorted array in ascending order:\n");
    for(i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}

```

7.3 INSERTION SORT

```

#include <stdio.h>

int main() {
    int arr[100], n, i, j, key;

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

```

```
printf("Enter %d elements:\n", n);
```

```
for(i = 0; i < n; i++) {  
    scanf("%d", &arr[i]);  
}
```

```
// Insertion Sort logic
```

```
for(i = 1; i < n; i++) {  
    key = arr[i];  
    j = i - 1;  
  
    while(j >= 0 && arr[j] > key) {  
        arr[j + 1] = arr[j];  
        j = j - 1;  
    }  
    arr[j + 1] = key;  
}
```

```
printf("\nSorted array in ascending order:\n");
```

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

```
    return 0;
}
```

7.4 Merge Sort

```
#include <stdio.h>

// Function to merge two halves
void merge(int arr[], int left, int mid, int right) {
    int i, j, k;
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int L[100], R[100];

    // Copy data to temporary arrays L[] and R[]
    for(i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for(j = 0; j < n2; j++)
```

```
R[j] = arr[mid + 1 + j];
```

```
i = 0; j = 0; k = left;
```

```
// Merge the temp arrays back into arr[]
```

```
while(i < n1 && j < n2) {
```

```
    if(L[i] <= R[j]) {
```

```
        arr[k] = L[i];
```

```
        i++;
```

```
    } else {
```

```
        arr[k] = R[j];
```

```
        j++;
```

```
    }
```

```
    k++;
```

```
}
```

```
// Copy remaining elements
```

```
while(i < n1) {
```

```
    arr[k] = L[i];
```

```
    i++; k++;
```

```
}
```

```
while(j < n2) {
```

```

        arr[k] = R[j];

        j++; k++;
    }
}

// Function to implement merge sort
void mergeSort(int arr[], int left, int right) {
    if(left < right) {
        int mid = (left + right) / 2;

        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

        merge(arr, left, mid, right);
    }
}

int main() {
    int arr[100], n, i;

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

```

```
printf("Enter %d elements:\n", n);  
for(i = 0; i < n; i++)  
    scanf("%d", &arr[i]);  
  
mergeSort(arr, 0, n - 1);  
  
printf("\nSorted array in ascending order:\n");  
for(i = 0; i < n; i++)  
    printf("%d ", arr[i]);  
  
return 0;  
}
```

7.5 QUICK SORT:

```
#include <stdio.h>  
  
// Function to swap two elements  
void swap(int *a, int *b) {  
    int temp = *a;  
    *a = *b;  
    *b = temp;  
}
```

```
}
```

```
// Partition function
```

```
int partition(int arr[], int low, int high) {
```

```
    int pivot = arr[high];
```

```
    int i = (low - 1);
```

```
    for (int j = low; j < high; j++) {
```

```
        if (arr[j] <= pivot) {
```

```
            i++;
```

```
            swap(&arr[i], &arr[j]);
```

```
        }
```

```
    }
```

```
    swap(&arr[i + 1], &arr[high]);
```

```
    return (i + 1);
```

```
}
```

```
// Quick Sort function
```

```
void quickSort(int arr[], int low, int high) {
```

```
    if (low < high) {
```

```
        int pi = partition(arr, low, high);
```

```
        quickSort(arr, low, pi - 1);
```

```
        quickSort(arr, pi + 1, high);
```

```
}  
}
```

```
int main() {  
    int arr[100], n, i;  
  
    printf("Enter number of elements: ");  
    scanf("%d", &n);  
  
    printf("Enter %d elements:\n", n);  
    for (i = 0; i < n; i++) {  
        scanf("%d", &arr[i]);  
    }  
  
    quickSort(arr, 0, n - 1);  
  
    printf("\nSorted array in ascending order:\n");  
    for (i = 0; i < n; i++) {  
        printf("%d ", arr[i]);  
    }  
  
    return 0;
```

```
}
```

EXPERIMENT 8: PROGRAM ON SEARCHING

Searching in BST:

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
// Structure of a node
```

```
struct node {
```

```
    int data;
```

```
    struct node *left, *right;
```

```
};
```

```
// Function to create a new node
```

```
struct node* newNode(int value) {
```

```
    struct node* temp = (struct node*)malloc(sizeof(struct node));
```

```
    temp->data = value;
```

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

```
    return temp;
}

// Function to insert a new node into BST
struct node* insert(struct node* root, int value) {
    if (root == NULL)
        return newNode(value);
    if (value < root->data)
        root->left = insert(root->left, value);
    else if (value > root->data)
        root->right = insert(root->right, value);
    return root;
}
```

```
// Function to search a key in BST
struct node* search(struct node* root, int key) {
    if (root == NULL || root->data == key)
        return root;
    if (key < root->data)
        return search(root->left, key);
    return search(root->right, key);
}
```

```
int main() {
    struct node* root = NULL;
```

```
root = insert(root, 50);
```

```
insert(root, 30);
```

```
insert(root, 70);
```

```
insert(root, 20);
```

```
insert(root, 40);
```

```
insert(root, 60);
```

```
insert(root, 80);
```

```
int key;
```

```
printf("Enter element to search: ");
```

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

```
struct node* result = search(root, key);
```

```
if (result != NULL)
```

```
    printf("Element %d found in BST.\n", key);
```

```
else
```

```
    printf("Element %d not found in BST.\n", key);
```

```
return 0;
```

```
}
```

8.2 Searching in Hash Table

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#define SIZE 7
```

```
struct node {
```

```
int data;

struct node* next;

};


struct node* hashTable[SIZE];


// Hash function
int hashFunction(int key) {
    return key % SIZE;
}


// Insert function
void insert(int key) {
    int index = hashFunction(key);
    struct node* newNode = (struct node*)malloc(sizeof(struct node));
    newNode->data = key;
    newNode->next = hashTable[index];
    hashTable[index] = newNode;
}


// Search function
int search(int key) {
    int index = hashFunction(key);
    struct node* temp = hashTable[index];
    while (temp != NULL) {
```

```
    if (temp->data == key)
        return 1; // Found
    temp = temp->next;
}
return 0; // Not Found
}
```

```
int main() {
    insert(14);
    insert(21);
    insert(28);
    insert(9);
    insert(15);

    int key;
    printf("Enter key to search: ");
    scanf("%d", &key);

    if (search(key))
        printf("Key %d found in hash table.\n", key);
    else
        printf("Key %d not found in hash table.\n", key);

    return 0;
}
```

Signature, Faculty

(Dr A. Brahma)