

**Practical File**

**Programming in JAVA Lab**

**(PCS 302)**

**B. Tech Fourth Semester**

**Session : 2024 - 25**

**Submitted to: Submitted by:**

**Mr. Parthak Mehra Soumya Bhakuni**

Assistant Professor B. Tech (CSE)

Computer Science & Engineering Section: A

Graphic Era Hill University Uni. Roll No: 2361519

Bhimtal Campus Class Roll. No. : 48

COLLEGE ROLL NO :23012292 EXAMINATION ROLL NO :2361519



THIS IS TO CERTIFY THAT **Ms. Soumya Bhakuni** HAS SATISFACTORILY COMPLETED ALL THE EXPERIMENTS IN THE LABORATORY OF THIS COLLEGE. THE COURSE OF THE EXPERIMENTS / TERM WORK IN **Programming in JAVA Lab (PCS 302)** IN PARTIAL FULFILMENT OF THE REQUIREMENT IN **FOURTH SEMESTER** OF **BACHELOR OF TECHNOLOGY (C.S.E.)** DEGREE COURSE PRESCRIBED BY THE GRAPHIC ERA HILL UNIVERSITY, BHIMTAL DURING THE YEAR **2024 - 25**.

CONCERNED FACULTY HEAD OF DEPARTMENT

NAME OF EXAMINER:

SIGNATURE OF EXAMINER:



**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Exp. No.** | **Date** | **Experiment** | **Page No.** | **Sign** |
| **1** |  | Write a C program to calculate the sum of all even elements in an array. |  |  |
| **2** |  | Write a C program to determine the union of two given arrays. |  |  |
| **3** |  | Write a C program to determine the intersection of two given arrays. |  |  |
| **4** |  | Write a C program to store N elements in an array and print the contents of the array in reverse order. |  |  |
| **5** |  | Write a C program to find the element with the maximum value from an array. |  |  |
| **6** |  | Write a C program to create a dynamic array. |  |  |
| **7** |  | Write a C program to Implementation Stack Using Array. |  |  |
| **8** |  | Write a C program to Implementation queue Using Array. |  |  |
| **9** |  | Write a C program to convert infix expression into postfix expression. |  |  |
| **10** |  | Write a C program to evaluate any postfix expression. |  |  |
| **11** |  | Write a C program to create and display a Singly Linked List. |  |  |
| **12** |  | Write a C program to insert a new node at the beginning, middle and end of a Singly Linked List. |  |  |
| **13** |  | Write a C program to insert a new node at any given position in a singly linked list. |  |  |
| **14** |  | Write a C program to search for an existing element in a singly linked list. |  |  |
| **15** |  | Write a C program that allows the user to input a key for searching in a singly linked list. The program should delete the node containing the key value and update the linked list accordingly. If the key is not found, display the message “Unsuccessful Search.” |  |  |
| **16** |  | Write a C program to merge two sorted singly linked lists into a single sorted linked list. |  |  |
| **17** |  | Write a C program to create and display a doubly linked list. |  |  |
| **18** |  | Write a C program to insert a node at the beginning, middle and end of a doubly linked list. |  |  |
| **19** |  | Write a C program to insert a new node at any given position in a doubly linked list. |  |  |
| **20** |  | Write a C program to search for an element in a doubly linked list and delete that element from the list. |  |  |
| **21** |  | Write a C program to create and display a Circular linked list. |  |  |
| **22** |  | Write a C program to insert a node at the beginning, middle and end of a Circular linked list. |  |  |
| **23** |  | Write a C program to insert a new node at any given position in a Circular linked list. |  |  |
| **24** |  | Write a C program to search for an element in a Circular linked list and then delete that element from the list. |  |  |
| **25** |  | Write a C program to insert string in linked list in alphabetical order. |  |  |
| **26** |  | Write a C program to remove duplicates from a linked list. |  |  |
| **27** |  | Write a C program to create and reorder a linked list placing all even-numbered nodes ahead of all odd-numbered nodes. |  |  |
| **28** |  | Write a C program to implement Stack Using linked List. |  |  |
| **29** |  | Write a C program to implement queue using double pointers. |  |  |
| **30** |  | Write a C program to implement Queue Using Linked List. |  |  |
| **31** |  | Write a C program to create Binary search tree and perform following operations on it. **i) Insert node ii) Delete node iii) Search node** |  |  |
| **32** |  | Write a C program to perform Inorder, Preorder and Postorder traversal on a Binary Search Tree. |  |  |
| **33** |  | Write a C program to delete a node from a Binary Search Tree. |  |  |
| **34** |  | Write a C program to calculate the height of a Binary Search Tree. |  |  |
| **35** |  | Write a C program to find the minimum and maximum values and total number of nodes in a Binary Search Tree. |  |  |
| **36** |  | Write a C program to check if a Binary Search Tree is balanced. |  |  |
| **37** |  | Write a C program to create a simple graph structure using adjacency lists. |  |  |
| **38** |  | Write a C program to sort an array using Bubble Sort technique |  |  |
| **39** |  | Write a C program to sort an array using Selection Sort technique |  |  |
| **40** |  | Write a C program to sort an array using Insertion Sort technique |  |  |
| **41** |  | Write a C program to sort an array using Merge Sort technique. |  |  |
| **42** |  | Write a C program to sort an array using Quick Sort technique. |  |  |
| **43** |  | Write a C program to implement Linear Search and Binary Search. |  |  |

**PROGRAM OBJECTIVE:** Write a C program to calculate the sum of all even elements in an array.

**PROGRAM CODE:**

#include <stdio.h>

int main() {

int n, i, sum = 0;

// Ask the user for the number of elements

printf("Enter the number of elements in the array: ");

scanf("%d", &n);

int arr[n];

// Read elements of the array from the user

printf("Enter the elements of the array:\n");

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

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

}

// Calculate the sum of all even elements

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

if (arr[i] % 2 == 0) {

sum += arr[i];

}

}

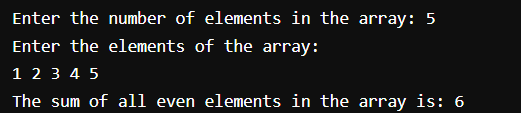
// Output the result

printf("The sum of all even elements in the array is: %d\n", sum);

return 0;

}

**OUTPUT:**



**PROGRAM OBJECTIVE:**Write a C program to determine the union of two given arrays.

**PROGRAM CODE:**

#include <stdio.h>

int isPresent(int arr[], int size, int element) { // Function to check if an element is present

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

if (arr[i] == element) {

return 1;}}

return 0;}

int main() {

int n1, n2;

// Read the size of the first array

printf("Enter the number of elements in the first array: ");

scanf("%d", &n1);

int arr1[n1];

// Read the elements of the first array

printf("Enter the elements of the first array:\n");

for (int i = 0; i < n1; i++) {

scanf("%d", &arr1[i]);

}

// Read the size of the second array

printf("Enter the number of elements in the second array: ");

scanf("%d", &n2);

int arr2[n2];

// Read the elements of the second array

printf("Enter the elements of the second array:\n");

for (int i = 0; i < n2; i++) {

scanf("%d", &arr2[i]);

} printf("Union of the two arrays:\n");

for (int i = 0; i < n1; i++) {

printf("%d ", arr1[i]);

}

for (int i = 0; i < n2; i++) {

if (!isPresent(arr1, n1, arr2[i])) {

printf("%d ", arr2[i]);

}

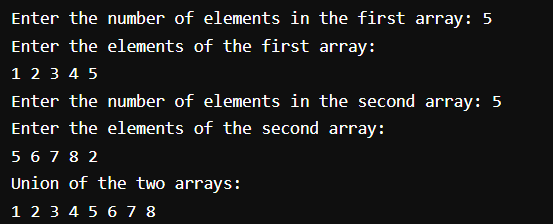
}

printf("\n");

return 0;

}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to determine the intersection of two given arrays.

**PROGRAM CODE:**

#include <stdio.h>

int isPresent(int arr[], int size, int element) {

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

if (arr[i] == element) {

return 1;}}

return 0;

}

int main() {

int n1, n2;

printf("Enter the number of elements in the first array: ");

scanf("%d", &n1);

int arr1[n1];

printf("Enter the elements of the first array:\n");

for (int i = 0; i < n1; i++) {

scanf("%d", &arr1[i]); }

printf("Enter the number of elements in the second array: ");

scanf("%d", &n2);

int arr2[n2];

printf("Enter the elements of the second array:\n");

for (int i = 0; i < n2; i++) {

scanf("%d", &arr2[i]); }

printf("Intersection of the two arrays:\n");

for (int i = 0; i < n1; i++) {

if (isPresent(arr2, n2, arr1[i])) {

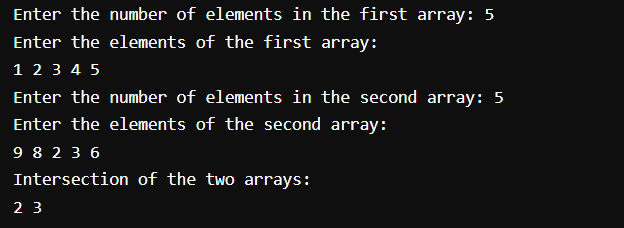
printf("%d ", arr1[i]); } }

printf("\n");

return 0;

}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to store N elements in an array and print the contents of the array in reverse order.

**PROGRAM CODE:**

#include <stdio.h>

int main() {

int n;

printf("Enter the number of elements in the array: ");

scanf("%d", &n); // Ask the user for the number of elements

int arr[n];

printf("Enter the elements of the array:\n");

for (int i = 0; i < n; i++) { // Read elements of the array from the user

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

}

printf("Array in reverse order:\n");

for (int i = n - 1; i >= 0; i--) { // Print the contents of the array in reverse order

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

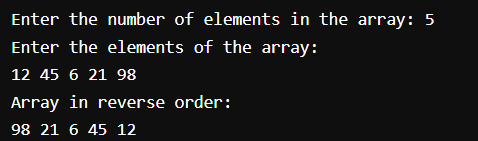
}

printf("\n");

return 0;

}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to find the element with the maximum value from an array.

**PROGRAM CODE:**

#include <stdio.h>

int main() {

int n, max;

printf("Enter the number of elements in the array: ");

scanf("%d", &n); // Ask the user for the number of elements

int arr[n];

printf("Enter the elements of the array:\n");

for (int i = 0; i < n; i++) { // Read elements of the array from the user

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

}

max = arr[0]; // Initialize max with the first element

// Find the maximum value in the array

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

if (arr[i] > max) {

max = arr[i];

}

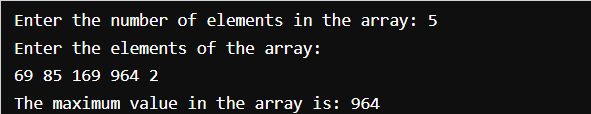
}

printf("The maximum value in the array is: %d\n", max);

return 0;

}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to create a dynamic array.

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

int main() {

int n, i;

int \*array;

printf("Enter the number of elements in the array: ");

scanf("%d", &n);

array = (int \*)malloc(n \* sizeof(int)); // Allocate memory

if (array == NULL) {

printf("Memory allocation failed.\n");

return 1;

}

printf("Enter the elements of the array:\n");

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

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

}

printf("The elements of the array are:\n");

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

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

}

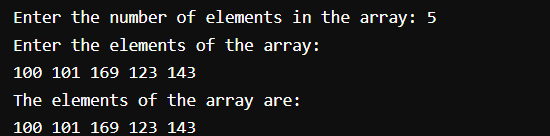
printf("\n");

free(array);

return 0;

}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to Implementation Stack Using Array.

**PROGRAM CODE:**

#include <stdio.h>

#define MAX 100

int stack[MAX];

int top = -1;

void push(int value) {

if (top == MAX - 1) {

printf("Stack Overflow! Cannot push %d\n", value);

} else {

stack[++top] = value;

printf("Pushed %d onto the stack.\n", value);}}

int pop() {

if (top == -1) {

printf("Stack Underflow! Stack is empty.\n");

return -1;

} else {

printf("Popped %d from the stack.\n", stack[top]);

return stack[top--];}}

void display() {

if (top == -1) {

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

} else {

printf("Stack elements are: ");

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

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

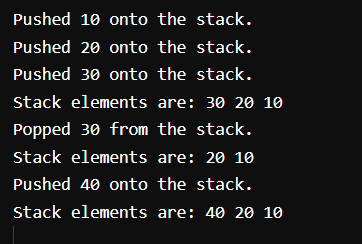
printf("\n");}}

int main() {

push(10); push(20); push(30); display(); pop(); display(); push(40); display();

return 0;}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to Implementation queue Using Array.

**PROGRAM CODE:**

#include <stdio.h>

#define SIZE 100

int queue[SIZE];

int front = -1;int rear = -1;

int isEmpty() { return front == -1 || front > rear;}

int isFull() { return rear == SIZE - 1;}

void enqueue(int value) {

if (isFull()) {

printf("Queue overflow! Cannot enqueue %d\n", value); return;}

if (front == -1) {

front = 0;}

queue[++rear] = value; printf("Enqueued: %d\n", value);}

int dequeue() {

if (isEmpty()) {

printf("Queue underflow! Cannot dequeue.\n"); return -1;}

int value = queue[front++];

if (front > rear) {

front = rear = -1;}

printf("Dequeued: %d\n", value);

return value;}

void display() {

if (isEmpty()) {

printf("Queue is empty.\n"); return;}

printf("Queue elements: ");

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

printf("%d ", queue[i]);}

printf("\n");}

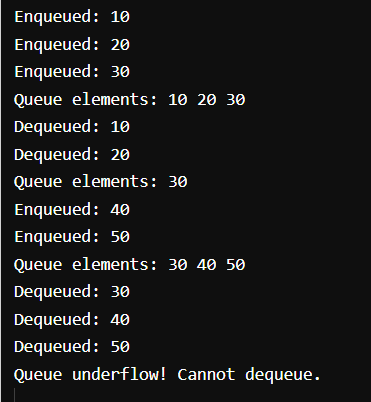
int main() {

enqueue(10); enqueue(20); enqueue(30); display(); dequeue(); dequeue();

display(); enqueue(40); enqueue(50); display(); dequeue(); dequeue(); dequeue(); dequeue();

return 0;}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to convert infix expression into postfix expression.

**PROGRAM CODE:**

#include <stdio.h>

#include <ctype.h>

#include <string.h>

#define MAX 100

char stack[MAX];

int top = -1;

void push(char c) {

if (top != MAX - 1) stack[++top] = c;}

char pop() {

return (top == -1) ? '\0' : stack[top--];}

char peek() {

return (top == -1) ? '\0' : stack[top];}

int precedence(char operator) {

switch (operator) {

case '+': case '-': return 1;

case '\*': case '/': return 2;

case '^': return 3;

default: return 0; }}

void infixToPostfix(char\* infix, char\* postfix) {

int i = 0, j = 0;

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

if (isalnum(infix[i])) {

postfix[j++] = infix[i];

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

push(infix[i]);

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

while (peek() != '(') postfix[j++] = pop();

pop();

} else {

while (precedence(peek()) >= precedence(infix[i])) postfix[j++] = pop();

push(infix[i]);

} i++; }

while (top != -1) postfix[j++] = pop(); postfix[j] = '\0';}

int main() {

char infix[MAX], postfix[MAX];

printf("Enter an infix expression: ");

scanf("%s", infix);

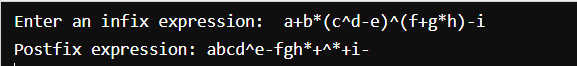
infixToPostfix(infix, postfix);

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

return 0;

}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to evaluate any postfix expression.

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#define MAX 100

typedef struct Stack {

int data[MAX];

int top;

} Stack;

void push(Stack \*stack, int value) {

stack->data[++stack->top] = value;}

int pop(Stack \*stack) {

return stack->data[stack->top--];}

int evaluatePostfix(char \*postfix) {

Stack stack = { .top = -1 };

int i = 0, val1, val2;

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

if (isdigit(postfix[i])) {

push(&stack, postfix[i] - '0');

} else {

val1 = pop(&stack);

val2 = pop(&stack);

switch (postfix[i]) {

case '+': push(&stack, val2 + val1); break;

case '-': push(&stack, val2 - val1); break;

case '\*': push(&stack, val2 \* val1); break;

case '/': push(&stack, val2 / val1); break;

} } i++; }

return pop(&stack);}

int main() {

char postfix[MAX];

printf("Enter a postfix expression: ");

scanf("%s", postfix);

printf("Result: %d\n", evaluatePostfix(postfix));

return 0;

}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to create and display a Singly Linked List.

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* createNode(int data) {

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

newNode->data = data;

newNode->next = NULL;

return newNode;

}

void displayList(struct Node\* head) {

struct Node\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL;

struct Node\* second = NULL;

struct Node\* third = NULL;

head = createNode(10);

second = createNode(20);

third = createNode(30);

head->next = second;

second->next = third;

printf("Singly Linked List: ");

displayList(head);

return 0;}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to insert a new node at the beginning, middle and end of a Singly Linked List.

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Node { int data; struct Node\* next;};

struct Node\* createNode(int data) {

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

newNode->data = data;

newNode->next = NULL; return newNode;}

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

struct Node\* newNode = createNode(data);

newNode->next = \*head;

\*head = newNode;}

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

struct Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode; return; }

struct Node\* temp = \*head;

while (temp->next != NULL) {

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

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

if (position <= 0) {

printf("Position must be greater than 0.\n"); return; }

struct Node\* newNode = createNode(data);

if (position == 1) {

newNode->next = \*head;

\*head = newNode; return; }

struct Node\* temp = \*head;

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

temp = temp->next; }

if (temp == NULL) {

printf("Position out of bounds.\n");

free(newNode); return; }

newNode->next = temp->next;

temp->next = newNode;}

void displayList(struct Node\* head) {

struct Node\* temp = head;

while (temp != NULL) {

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

temp = temp->next; }

printf("NULL\n");}

int main() {

struct Node\* head = NULL;

insertAtBeginning(&head, 10); insertAtEnd(&head, 20); insertAtMiddle(&head, 15, 2);

printf("Linked List: "); displayList(head); return 0;}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to insert a new node at any given position in a singly linked list.

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;};

struct Node\* createNode(int data) {

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

newNode->data = data;

newNode->next = NULL; return newNode;}

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

if (position < 1) {

printf("Position must be greater than or equal to 1.\n"); return; }

struct Node\* newNode = createNode(data);

if (position == 1) {

newNode->next = \*head;

\*head = newNode; return; }

struct Node\* temp = \*head;

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

temp = temp->next; }

if (temp == NULL) {

printf("Position out of bounds.\n");

free(newNode); return; }

newNode->next = temp->next;

temp->next = newNode;}

void displayList(struct Node\* head) {

struct Node\* temp = head;

while (temp != NULL) {

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

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

int main() {

struct Node\* head = NULL;

insertAtPosition(&head, 10, 1); insertAtPosition(&head, 20, 2); insertAtPosition(&head, 15, 2);

printf("Linked List: "); displayList(head); return 0;}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to search for an existing element in a singly linked list

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Node { int data; struct Node\* next;};

struct Node\* createNode(int data) {

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

newNode->data = data;

newNode->next = NULL; return newNode;}

int searchElement(struct Node\* head, int value) {

struct Node\* temp = head; int position = 1;

while (temp != NULL) {

if (temp->data == value) { return position;

} temp = temp->next; position++; }

return -1; }

void displayList(struct Node\* head) {

struct Node\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL;

struct Node\* temp = NULL;

int n, value, position;

printf("Enter the number of elements in the linked list: ");

scanf("%d", &n);

if (n > 0) {

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

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

int data;

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

scanf("%d", &data);

struct Node\* newNode = createNode(data);

if (head == NULL) {

head = newNode;

temp = head;

} else {

temp->next = newNode;

temp = newNode;

}

}

printf("Enter the value to search: ");

scanf("%d", &value);

position = searchElement(head, value);

if (position != -1) {

printf("Element %d found at position %d.\n", value, position);

} else {

printf("Element %d not found in the list.\n", value);

}

printf("Linked List: ");

displayList(head);

} else {

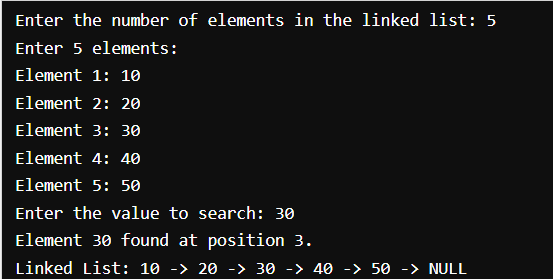
printf("The linked list cannot be empty.\n");

}

return 0;

}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program that allows the user to input a key for searching in a singly linked list. The program should delete the node containing the key value and update the linked list accordingly. If the key is not found, display the message “Unsuccessful Search.”

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data; struct Node\* next;};

struct Node\* createNode(int data) {

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

newNode->data = data;

newNode->next = NULL; return newNode;}

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

struct Node\* temp = \*head;

struct Node\* prev = NULL;

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

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

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

prev = temp;

temp = temp->next; }

if (temp == NULL) {

printf("Unsuccessful Search.\n"); return; }

prev->next = temp->next; free(temp);}

void displayList(struct Node\* head) {

struct Node\* temp = head;

while (temp != NULL) {

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

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

int main() {

struct Node\* head = NULL;

struct Node\* temp = NULL;

int n, key, position;

printf("Enter the number of elements in the linked list: ");

scanf("%d", &n);

if (n > 0) {

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

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

int data;

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

scanf("%d", &data);

struct Node\* newNode = createNode(data);

if (head == NULL) {

head = newNode;

temp = head;

} else {

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

printf("Enter the key value to delete: ");

scanf("%d", &key);

deleteNode(&head, key);

printf("Updated Linked List: ");

displayList(head);

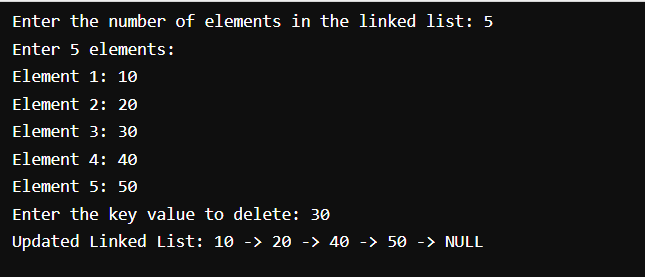
} else {

printf("The linked list cannot be empty.\n"); }

return 0;

}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to merge two sorted singly linked lists into a single sorted linked list.

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Node { int data; struct Node\* next;};

struct Node\* createNode(int data) {

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

newNode->data = data; newNode->next = NULL; return newNode;}

void displayList(struct Node\* head) {

while (head) {

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

printf("NULL\n");}

struct Node\* mergeLists(struct Node\* list1, struct Node\* list2) {

struct Node\* merged = NULL, \*\*tail = &merged;

while (list1 && list2) {

if (list1->data <= list2->data) {

\*tail = list1; list1 = list1->next; } else {

\*tail = list2; list2 = list2->next; } tail = &((\*tail)->next); }

\*tail = (list1) ? list1 : list2; return merged;}

int main() {

struct Node \*list1 = NULL, \*list2 = NULL, \*temp; int n, data;

printf("Enter number of elements for list1: "); scanf("%d", &n);

while (n--) {

printf("Element: "); scanf("%d", &data);

struct Node\* newNode = createNode(data);

if (!list1) list1 = newNode;

else temp->next = newNode; temp = newNode; } temp = NULL;

printf("Enter number of elements for list2: "); scanf("%d", &n);

while (n--) {

printf("Element: "); scanf("%d", &data);

struct Node\* newNode = createNode(data);

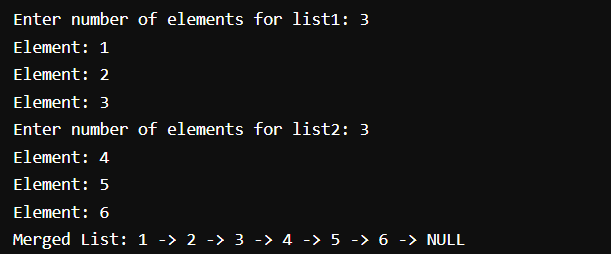
if (!list2) list2 = newNode;

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

struct Node\* merged = mergeLists(list1, list2);

printf("Merged List: "); displayList(merged); return 0;}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to create and display a doubly linked list.

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* prev;

struct Node\* next;

};

struct Node\* createNode(int data) {

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

newNode->data = data;

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

return newNode;

}

void displayList(struct Node\* head) {

while (head) {

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

head = head->next;

}

printf("NULL\n");

}

int main() {

struct Node \*head = NULL, \*temp = NULL;

head = createNode(10);

temp = head;

temp->next = createNode(20);

temp->next->prev = temp;

temp = temp->next;

temp->next = createNode(30);

temp->next->prev = temp;

temp = temp->next;

temp->next = createNode(40);

temp->next->prev = temp;

displayList(head);

return 0;

}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to insert a node at the beginning, middle and end of a doubly linked list.

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data; struct Node\* prev; struct Node\* next;};

void displayList(struct Node\* head) {

while (head) {

printf("%d <-> ", head->data); head = head->next; }

printf("NULL\n");}

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

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

newNode->data = data; newNode->prev = NULL;

newNode->next = \*head;

if (\*head) (\*head)->prev = newNode; \*head = newNode;}

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

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

newNode->data = data; newNode->next = NULL;

if (!\*head) {

newNode->prev = NULL; \*head = newNode;

} else {

struct Node\* temp = \*head;

while (temp->next) temp = temp->next; temp->next = newNode;

newNode->prev = temp; }}

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

if (position <= 0) return;

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

newNode->data = data;

struct Node\* temp = \*head;

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

if (temp) {

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

if (temp->prev) temp->prev->next = newNode; temp->prev = newNode;

} else { insertAtEnd(head, data); }}

int main() {

struct Node\* head = NULL;

insertAtBeginning(&head, 10); insertAtEnd(&head, 20); insertAtEnd(&head, 30);

insertAtMiddle(&head, 15, 2); printf("Doubly Linked List: "); displayList(head)

return 0;}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to insert a new node at any given position in a doubly linked list.

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data; struct Node\* prev; struct Node\* next;};

void displayList(struct Node\* head) {

while (head) {

printf("%d <-> ", head->data); head = head->next; }

printf("NULL\n");}

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

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

newNode->data = data;

if (position <= 0) return;

if (position == 1) {

newNode->prev = NULL;

newNode->next = \*head;

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

\*head = newNode;

} else {

struct Node\* temp = \*head;

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

temp = temp->next;

}

if (temp) { // Insert in the middle or at the end

newNode->prev = temp;

newNode->next = temp->next;

if (temp->next) temp->next->prev = newNode;

temp->next = newNode;

} else {

free(newNode); printf("Position out of range\n"); } }}

int main() {

struct Node\* head = NULL;

insertAtPosition(&head, 10, 1); insertAtPosition(&head, 20, 2);

insertAtPosition(&head, 30, 3); insertAtPosition(&head, 15, 2); printf("Doubly Linked List: "); displayList(head); return 0;}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to search for an element in a doubly linked list and delete that element from the list.

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data; struct Node\* prev; struct Node\* next;};

void displayList(struct Node\* head) {

while (head) {

printf("%d <-> ", head->data); head = head->next; }

printf("NULL\n");}

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

if (\*head == NULL) return;

struct Node\* temp = \*head;

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

temp = temp->next; }

if (temp == NULL) {

printf("Element %d not found in the list.\n", key); return; }

if (\*head == temp) { \*head = temp->next; }

if (temp->next) { temp->next->prev = temp->prev; }

if (temp->prev) { temp->prev->next = temp->next; } free(temp);

printf("Element %d deleted from the list.\n", key);}

int main() {

struct Node\* head = NULL;

struct Node\* node1 = (struct Node\*)malloc(sizeof(struct Node));

node1->data = 10; node1->prev = NULL; node1->next = NULL;

head = node1;

struct Node\* node2 = (struct Node\*)malloc(sizeof(struct Node));

node2->data = 20; node2->prev = node1; node2->next = NULL;

node1->next = node2;

struct Node\* node3 = (struct Node\*)malloc(sizeof(struct Node));

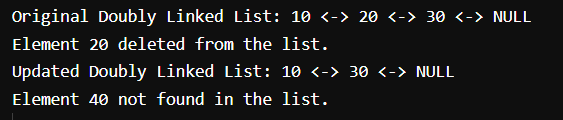
node3->data = 30; node3->prev = node2; node3->next = NULL;

node2->next = node3;

printf("Original Doubly Linked List: "); displayList(head); deleteNode(&head, 20);

printf("Updated Doubly Linked List: "); displayList(head); deleteNode(&head, 40); return 0;}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to create and display a Circular linked list.

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data; struct Node\* next;};

void displayList(struct Node\* head) {

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

struct Node\* temp = head;

do {

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

} while (temp != head);

printf("NULL\n");}

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

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

struct Node\* temp = \*head;

newNode->data = data;

if (\*head == NULL) {

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

} else {

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

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

int main() {

struct Node\* head = NULL;

insertAtEnd(&head, 10);

insertAtEnd(&head, 20);

insertAtEnd(&head, 30);

printf("Circular Linked List: ");

displayList(head);

return 0;

}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to insert a node at the beginning, middle and end of a Circular linked list.

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data; struct Node\* next;};

void displayList(struct Node\* head) {

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

struct Node\* temp = head;

do {

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

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

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

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

newNode->data = 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; }}

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

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

newNode->data = 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; }}

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

if (\*head == NULL || position <= 1) { insertAtBeginning(head, data);return; }

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

newNode->data = data; struct Node\* temp = \*head;

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

if (temp == NULL) { insertAtEnd(head, data); return; }

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

int main() {

struct Node\* head = NULL;

insertAtEnd(&head, 10); insertAtEnd(&head, 20); insertAtEnd(&head, 30);

insertAtBeginning(&head, 5); insertAtMiddle(&head, 15, 3); printf("Circular Linked List: "); displayList(head); return 0;}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to insert a new node at any given position in a Circular linked list.

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Node { int data; struct Node\* next;};

void displayList(struct Node\* head) {

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

struct Node\* temp = head;

do {

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

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

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

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

newNode->data = data; if (\*head == NULL) {

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

if (position == 1) { // Insert at the beginning

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

{ temp = temp->next; }

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

return; }

struct Node\* temp = \*head; int count = 1;

while (temp->next != \*head && count < position - 1) {

temp = temp->next; count++; }

if (count == position - 1) { // Insert in the middle or at the end

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

} else {

printf("Position out of range\n"); free(newNode); }}

int main() {

struct Node\* head = NULL;

insertAtPosition(&head, 10, 1); insertAtPosition(&head, 20, 2);

insertAtPosition(&head, 30, 3); insertAtPosition(&head, 15, 2);

insertAtPosition(&head, 5, 1); printf("Circular Linked List: "); displayList(head);

return 0;}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to search for an element in a Circular linked list and then delete that element from the list.

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Node { int data; struct Node\* next;};

void displayList(struct Node\* head) {

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

struct Node\* temp = head;

do {

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

printf("NULL\n");}void deleteNode(struct Node\*\* head, int key) {

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

struct Node \*temp = \*head, \*prev = NULL;

if (temp->data == key) { if (temp->next == \*head) { free(temp);

\*head = NULL; printf("Node with value %d deleted\n", key); return;}

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

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

printf("Node with value %d deleted\n", key); return; }

prev = temp; temp = temp->next;

while (temp != \*head) { if (temp->data == key) {

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

printf("Node with value %d deleted\n", key); return; }

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

printf("Node with value %d not found in the list\n", key);}

int main() { struct Node\* head = NULL;

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

newNode->data = 10; newNode->next = newNode; head = newNode;

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

newNode->data = 20; newNode->next = head; head->next = newNode;

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

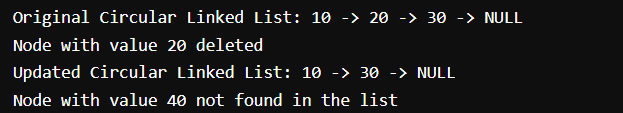
newNode->data = 30; newNode->next = head; head->next->next = newNode;

printf("Original Circular Linked List: "); displayList(head);

deleteNode(&head, 20); printf("Updated Circular Linked List: "); displayList(head);

deleteNode(&head, 40); return 0;}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to insert string in linked list in alphabetical order.

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

struct Node { char data[100]; struct Node\* next;};

void insertInOrder(struct Node\*\* head, char\* str) {

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

struct Node\* temp = \*head; strcpy(newNode->data, str); newNode->next = NULL;

if (\*head == NULL || strcmp((\*head)->data, str) > 0) {

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

while (temp->next && strcmp(temp->next->data, str) < 0) temp = temp->next;

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

}void display(struct Node\* head) {

while (head) {

printf("%s -> ", head->data);

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

int main() {

struct Node\* head = NULL; char str[100];

for (int i = 0; i < 3; i++) {

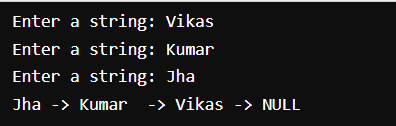
printf("Enter a string: "); fgets(str, sizeof(str), stdin);

str[strcspn(str, "\n")] = 0; insertInOrder(&head, str); }

display(head); return 0;

}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to remove duplicates from a linked list.

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data; struct Node\* next;};

void removeDuplicates(struct Node\* head) {

struct Node\* current = head;

struct Node\* prev = NULL;

struct Node\* temp = NULL;

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

prev = current; temp = current->next;

while (temp != NULL) { if (current->data == temp->data) {

prev->next = temp->next;

free(temp);

temp = prev->next;

} else {

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

current = current->next; }}

void displayList(struct Node\* head) {

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

struct Node\* temp = head;

while (temp != NULL) {

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

printf("NULL\n");}

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

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

struct Node\* temp = \*head;

newNode->data = data;

newNode->next = NULL;

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

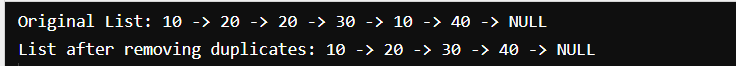
while (temp->next != NULL) { temp = temp->next; } temp->next = newNode;}

int main() { struct Node\* head = NULL;

insertAtEnd(&head, 10); insertAtEnd(&head, 20); insertAtEnd(&head, 20); insertAtEnd(&head, 30);

insertAtEnd(&head, 10); insertAtEnd(&head, 40); printf("Original List: "); displayList(head); removeDuplicates(head); printf("List after removing duplicates: "); displayList(head); return 0;}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to create and reorder a linked list placing all even-numbered nodes ahead of all odd-numbered nodes.

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

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

if (\*head == NULL) return;

struct Node \*evenHead = NULL, \*oddHead = NULL;

struct Node \*evenTail = NULL, \*oddTail = NULL; struct Node \*current = \*head;

while (current != NULL) {

if (current->data % 2 == 0) {

if (evenHead == NULL) { evenHead = current; evenTail = evenHead;

} else { evenTail->next = current; evenTail = evenTail->next; }

} else {

if (oddHead == NULL) { oddHead = current; oddTail = oddHead; }

else oddTail->next = current; oddTail = oddTail->next; } }

current = current->next; }

if (evenTail != NULL) evenTail->next = NULL;

if (oddTail != NULL) oddTail->next = NULL;

if (evenHead != NULL) { \*head = evenHead;

if (evenTail != NULL) evenTail->next = oddHead; } else {

\*head = oddHead; }}

void displayList(struct Node\* head) { if (head == NULL) {

printf("List is empty\n"); return; }

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

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

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

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

struct Node\* temp = \*head; newNode->data = data; newNode->next = NULL;

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

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

temp->next = newNode;}

int main() {

struct Node\* head = NULL;

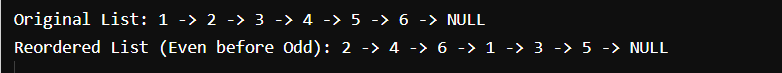
insertAtEnd(&head, 1); insertAtEnd(&head, 2); insertAtEnd(&head, 3); insertAtEnd(&head, 4);

insertAtEnd(&head, 5); insertAtEnd(&head, 6); printf("Original List: "); displayList(head);

reorderList(&head); printf("Reordered List (Even before Odd): "); displayList(head); return 0;

}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to implement Stack Using linked List.

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Node { int data; struct Node\* next;};

struct Stack { struct Node\* top;};

void initStack(struct Stack\* stack) { stack->top = NULL;}

int isEmpty(struct Stack\* stack) { return stack->top == NULL;}

void push(struct Stack\* stack, int data) {

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

newNode->data = data; newNode->next = stack->top; stack->top = newNode;

printf("%d pushed onto stack\n", data);}

int pop(struct Stack\* stack) {

if (isEmpty(stack)) { printf("Stack underflow\n"); return -1; }

struct Node\* temp = stack->top; int poppedValue = temp->data;

stack->top = stack->top->next; free(temp); return poppedValue;}

int peek(struct Stack\* stack) { if (isEmpty(stack)) { printf("Stack is empty\n"); return -1;

} return stack->top>data;}

void display(struct Stack\* stack) {

if (isEmpty(stack)) { printf("Stack is empty\n"); return; }

struct Node\* temp = stack->top;

printf("Stack elements: ");

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

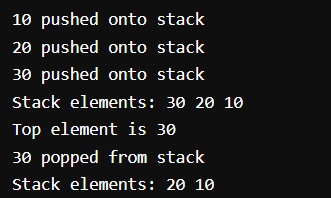
printf("\n");}

int main() { struct Stack stack;

initStack(&stack); push(&stack, 10); push(&stack, 20); push(&stack, 30); display(&stack);

printf("Top element is %d\n", peek(&stack)); printf("%d popped from stack\n", pop(&stack)); display(&stack); return 0;}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to implement queue using double pointers.

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Node { int data; struct Node\* next;};

struct Queue { struct Node\* front; struct Node\* rear;};

void initQueue(struct Queue\*\* queue) {

\*queue = (struct Queue\*)malloc(sizeof(struct Queue));

(\*queue)->front = (\*queue)->rear = NULL;}

int isEmpty(struct Queue\* queue) { return queue->front == NULL;}

void enqueue(struct Queue\* queue, int data) {

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

newNode->data = data; newNode->next = NULL;

if (queue->rear == NULL) { queue->front = queue->rear = newNode;

return; }

queue->rear->next = newNode; queue->rear = newNode;

printf("%d enqueued to queue\n", data);}

int dequeue(struct Queue\* queue) {

if (isEmpty(queue)) { printf("Queue undrflow\n"); return -1; }

struct Node\* temp = queue->front;

int dequeuedValue = temp->data;

queue->front = queue->front->next;

if (queue->front == NULL) { queue->rear = NULL; }

free(temp); return dequeuedValue;}

int front(struct Queue\* queue) {

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

return queue->front->data;}

void displayQueue(struct Queue\* queue) {

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

struct Node\* temp = queue->front;

printf("Queue elements: ") while (temp != NULL)

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

printf("\n");}

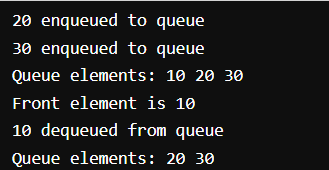
int main() {

struct Queue\* queue; initQueue(&queue);

enqueue(queue, 10); enqueue(queue, 20); enqueue(queue, 30); displayQueue(queue);

printf("Front element is %d\n", front(queue)); printf("%d dequeued from queue\n", dequeue(queue)); displayQueue(queue); return 0;}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to implement Queue Using Linked List.

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Node { int data; struct Node\* next;};

struct Queue { struct Node\* front; struct Node\* rear;};

void initQueue(struct Queue\* queue) { queue->front = queue->rear = NULL;}

int isEmpty(struct Queue\* queue) { return queue->front == NULL;}

void enqueue(struct Queue\* queue, int data) {

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

newNode->data = data; newNode->next = NULL; if (queue->rear == NULL) {

queue->front = queue->rear = newNode; return; }

queue->rear->next = newNode; queue->rear = newNode; printf("%d enqueued to queue\n", data);}

int dequeue(struct Queue\* queue) {

if (isEmpty(queue)) { printf("Queue underflow\n"); return -1; }

struct Node\* temp = queue->front; int dequeuedValue = temp->data;

queue->front = queue->front->next;

if (queue->front == NULL) { queue->rear = NULL; } free(temp);

return dequeuedValue;}

int front(struct Queue\* queue) {

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

return queue->front->data;}

void displayQueue(struct Queue\* queue) {

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

struct Node\* temp = queue->front;

printf("Queue elements: ");

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

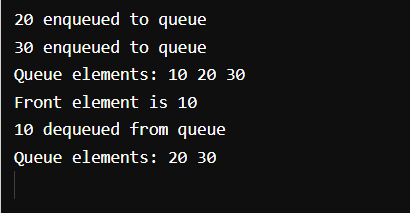
printf("\n");}

int main() { struct Queue queue;

initQueue(&queue); enqueue(&queue, 10); enqueue(&queue, 20); enqueue(&queue, 30); displayQueue(&queue); printf("Front element is %d\n", front(&queue));

printf("%d dequeued from queue\n", dequeue(&queue)); displayQueue(&queue); return 0;}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to create Binary search tree and perform following operations on it.

1. Insert node ii) Delete node iii) Search node.

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Node { int data; struct Node\* left; struct Node\* right; };

struct Node\* newNode(int data) {

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

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

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

if (root == NULL) { return newNode(data);} if (data < root->data) {

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

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

struct Node\* minValueNoe(struct Node\* node) {

struct Node\* current = node; while (current && current->lft != NULL) {

current = current->left; } return current;}

struct Node\* delete(struct Node\* root, int data) {

if (root == NULL) { return root;} if (data < root->data) {

root->left = delete(root->left, data); } else if (data > root->data) {

root->right = delete(root->right, data); } else { if (root->left == NULL) {

struct Node\* temp = root->right;

free(root); return temp;

} else if (root->right == NULL) {

struct Node\* temp = root->left; free(root); return temp;}

struct Node\* temp = minValueNode(root->right);

root->data = temp->data; root->right = delete(root->right, temp->data);}

return root;}

struct Node\* search(struct Node\* root, int data) {

if (root == NULL || root->data == data) { return root;}

if (data < root->data) { return search(root->left, data);}

return search(root->right, data);}

void inorder(struct Node\* root) {

if (root != NULL) { inorder(root->left); printf("%d ", root->data); inorder(root->right);}}

int main() {

struct Node\* root = NULL;

root = insert(root, 50); root = insert(root, 30); root = insert(root, 20); root = insert(root, 40);

root = insert(root, 70); root = insert(root, 60); root = insert(root, 80); printf("Inorder Traversal after insertions: "); inorder(root); printf("\n"); root = elete(root, 20);

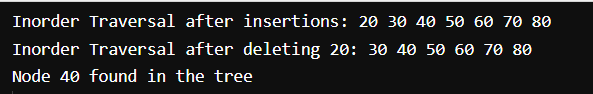
printf("Inorder Traversal after deleting 20: "); inorder(root); printf("\n");

int searchValue = 40; struct Node\* result = search(root, searchValue);

if (result != NULL) { printf("Node %d found in the tree\n", searchValue); } else {

printf("Node %d not found in the tree\n", searchValue); } return 0;}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to perform Inorder, Preorder and Postorder traversal on a Binary Search Tree.

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data; struct Node\* left; struct Node\* right;};

struct Node\* newNode(int data) {

struct Node\* node = (strct Node\*)malloc(sizeof(struct Node));

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

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

if (root == NULL) { return newNode(data); }

if (data < root->data) {

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

} else if (data > root->data) {

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

} return root;}

void inorder(struct Node\* root) {

if (root != NULL) {

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

void preorder(struct Node\* root) {

if (root != NULL) {

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

void postorder(struct Node\* root) {

if (root != NULL) {

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

int main() { struct Node\* root = NULL;

root = insert(root, 50); root = insert(root, 30); root = insert(root, 20); root = insert(root, 40);

root = insert(root, 70); root = insert(root, 60); root = insert(root, 80); printf("Inorder Traversal: ");

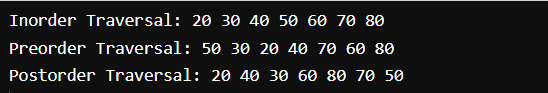
inorder(root); printf("\n"); printf("Preorder Traversal: "); preorder(root); printf("\n");

printf("Postorder Traversal: ");

postorder(root); printf("\n"); return 0;

}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to delete a node from a Binary Search Tree.

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Node { int key; struct Node\* left; struct Node\* right;};

struct Node\* getSuccessor(struct Node\* curr) {

curr = curr->right; while (curr != NULL && curr->left != NULL)

curr = curr->left; return curr;}

struct Node\* delNode(struct Node\* root, int x) {

if (root == NULL) return root;

if (root->key > x) root->left = delNode(root->left, x);

else if (root->key < x) root->right = delNode(root->right, x);

else {

if (root->left == NULL) { struct Node\* temp = root->right;

free(root); return temp; }

if (root->right == NULL) {

struct Node\* temp = root->left; free(root); return temp; }

struct Node\* succ = getSuccessor(root); root->key = succ->key;

root->right = delNode(root->right, succ->key); } return root;}

struct Node\* createNode(int key) {

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

newNode->key = key;newNode->left = newNode->right = NULL;

return newNode;}

void inorder(struct Node\* root) {

if (root != NULL) { inorder(root->left); printf("%d ", root->key); inorder(root->right); }}

int main() {

struct Node\* root = createNode(10); root->left = createNode(5); root->right = createNode(15);

root->right->left = createNode(12); root->right->right = createNode(18); int x = 15;

root = delNode(root, x); inorder(root); return 0;}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to calculate the height of a Binary Search Tree.

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node \*left;

struct Node \*right;

};

struct Node\* newNode(int data) {

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

node->data = data;

node->left = NULL;

node->right = NULL;

return node;

}

int height(struct Node\* root) {

if (root == NULL) {

return 0; // Height of an empty tree is 0

} else {

// Calculate the height of left and right subtrees

int leftHeight = height(root->left);

int rightHeight = height(root->right);

return (leftHeight > rightHeight) ? leftHeight + 1 : rightHeight + 1;

}

}

int main() {

struct Node\* root = newNode(10);

root->left = newNode(5);

root->right = newNode(15);

root->left->left = newNode(3);

root->left->right = newNode(7);

int treeHeight = height(root);

printf("Height of the BST: %d\n", treeHeight);

return 0;

}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to find the minimum and maximum values and total number of nodes in a Binary Search Tree.

**PROGRAM CODE:**

#include <limits.h>

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node \*left, \*right;

};

struct Node\* newNode(int data)

{

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

node->data = data;

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

return node;

}

int findMax(struct Node\* root)

{

if (root == NULL)

return INT\_MIN;

int res = root->data;

int lres = findMax(root->left);

int rres = findMax(root->right);

if (lres > res)

res = lres;

if (rres > res)

res = rres;

return res;

}

int findMin(struct Node\* root)

{ if (root == NULL) return INT\_MAX;

int res = root->data;

int lres = findMin(root->left);

int rres = findMin(root->right);

if (lres < res) res = lres; if (rres < res) res = rres; return res;}

nt main(void)

{

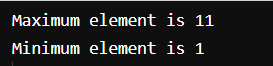
struct Node\* root = newNode(2); root->left = newNode(7); root->right = newNode(5);

root->left->right = newNode(6); root->left->right->left = newNode(1); root->left->right->right = newNode(11); root->right->right = newNode(9); root->right->right->left = newNode(4);

printf("Maximum element is %d \n", findMax(root)); printf("Minimum element is %d \n", findMin(root));

return 0;}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to check if a Binary Search Tree is balanced.

**PROGRAM CODE:**

#include <stdio.h>

#include <stdbool.h>

struct Node {

int data;

struct Node\* left;

struct Node\* right;};

struct Node\* newNode(int data) {

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

node->data = data;

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

return node;}

int height(struct Node\* root) {

if (root == NULL)

return 0;

int leftHeight = height(root->left);

int rightHeight = height(root->right);

return (leftHeight > rightHeight ? leftHeight : rightHeight) + 1;}

bool isBalanced(struct Node\* root) {

if (root == NULL)

return true;

int leftHeight = height(root->left);

int rightHeight = height(root->right);

if (abs(leftHeight - rightHeight) > 1)

return false;

return isBalanced(root->left) && isBalanced(root->right);}

int main() {

struct Node\* root = newNode(10);

root->left = newNode(5);

root->right = newNode(20);

root->left->left = newNode(3);

root->left->right = newNode(7);

root->right->right = newNode(30);

if (isBalanced(root))

printf("The tree is balanced\n");

else

printf("The tree is not balanced\n");

return 0;}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to sort an array using Bubble Sort technique.

**PROGRAM CODE:**

#include <stdio.h>

void bubbleSort(int arr[], int n) {

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

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

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

// Swap arr[j] and arr[j+1]

int temp = arr[j];

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

arr[j+1] = temp;

}

}

}

}

void printArray(int arr[], int n) {

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

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

}

printf("\n");

}

int main() {

int arr[] = {64, 25, 12, 22, 11};

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

printf("Original array: ");

printArray(arr, n);

bubbleSort(arr, n);

printf("Sorted array: ");

printArray(arr, n);

return 0;

}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to sort an array using Selection Sort technique.

**PROGRAM CODE:**

#include <stdio.h>

void selectionSort(int arr[], int n) {

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

// Find the minimum element in the unsorted portion

int minIndex = i;

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

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

if (minIndex != i) {

int temp = arr[i];

arr[i] = arr[minIndex];

arr[minIndex] = temp;

}

}

}

void printArray(int arr[], int n) {

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

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

}

printf("\n");

}

int main() {

int arr[] = {64, 25, 12, 22, 11};

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

printf("Original array: ");

printArray(arr, n);

selectionSort(arr, n);

printf("Sorted array: ");

printArray(arr, n);

return 0;

}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to sort an array using Insertion Sort technique.

**PROGRAM CODE:**

#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--;

}

arr[j + 1] = key;

}

}

void printArray(int arr[], int n) {

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

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

}

printf("\n");

}

int main() {

int n;

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

scanf("%d", &n);

int arr[n];

printf("Enter the elements of the array:\n");

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

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

}

insertionSort(arr, n);

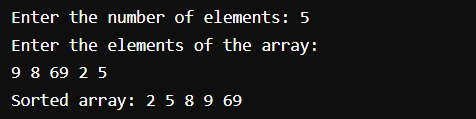
printf("Sorted array: ");

printArray(arr, n);

return 0;

}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to sort an array using Merge Sort technique.

**PROGRAM CODE:**

#include <stdio.h>

void merge(int arr[], int left, int mid, int right) {

int n1 = mid - left + 1;

int n2 = right - mid;

int L[n1], R[n2];

for (int i = 0; i < n1; i++)

L[i] = arr[left + i];

for (int j = 0; j < n2; j++)

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

int i = 0, j = 0, k = left;

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

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

arr[k] = L[i];

i++; } else { arr[k] = R[j]; j++; } k++; }

while (i < n1) {

arr[k] = L[i]; i++; k++; }

while (j < n2) { arr[k] = R[j]; j++; k++; }}

void mergeSort(int arr[], int left, int right) {

if (left < right) { int mid = left + (right - left) / 2;

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

void printArray(int arr[], int n) {

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

int main() {

int n; printf("Enter the number of elements: ");

scanf("%d", &n); int arr[n];

printf("Enter the elements of the array:\n");

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

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

}

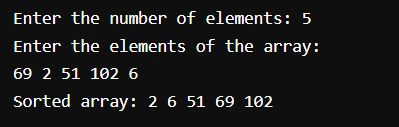
mergeSort(arr, 0, n - 1); printf("Sorted array: ");

printArray(arr, n);

return 0;

}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to sort an array using Quick Sort technique.

**PROGRAM CODE:**

#include <stdio.h>

void swap(int\* a, int\* b) {

int temp = \*a;

\*a = \*b;

\*b = temp;}

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;}

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);}}

void printArray(int arr[], int n) {

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

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

}

printf("\n");

}

int main() {

int n;

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

scanf("%d", &n);

int arr[n];

printf("Enter the elements of the array:\n");

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

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

}

quickSort(arr, 0, n - 1);

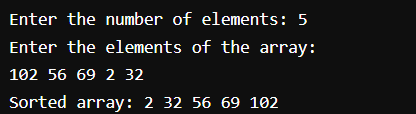
printf("Sorted array: ");

printArray(arr, n);

return 0;

}

**OUTPUT:**



**PROGRAM OBJECTIVE:** Write a C program to implement Linear Search and Binary Search.

**PROGRAM CODE:**

#include <stdio.h>

int linearSearch(int arr[], int n, int key) {

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

if (arr[i] == key) return i;}

return -1;}

int binarySearch(int arr[], int low, int high, int key) {

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] == key) return mid;

else if (arr[mid] < key) low = mid + 1;

else high = mid - 1;} return -1;}

void bubbleSort(int arr[], int n) {

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

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

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

int temp = arr[j];

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

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

void printResult(const char\* searchType, int result) {

if (result != -1) printf("%s: Element found at index %d\n", searchType, result);

else printf("%s: Element not found\n", searchType);}

void performSearches(int arr[], int n, int key) {

printResult("Linear Search", linearSearch(arr, n, key));

bubbleSort(arr, n);

printResult("Binary Search", binarySearch(arr, 0, n - 1, key));}

int main() {

int arr[] = {10, 23, 5, 17, 9};

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

int key = 17;

printf("Array elements: ");

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

printf("\nElement to search: %d\n", key);

performSearches(arr, n, key);

return 0;

}

**OUTPUT:**

