1. **Write a program to sort a given list of numbers in ascending order and print the original list of numbers with the sorted list using function.**

#include <stdio.h>

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

int temp;

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

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

if (arr[i] > arr[j]) {

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

}

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 %d elements: ", n);

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

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

}

printf("Original list: ");

printArray(arr, n);

sortArray(arr, n);

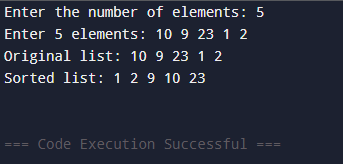
printf("Sorted list: ");

printArray(arr, n);

return 0;

}

**OUTPUT:**



1. **Write a program to calculate addition, subtraction, and multiplication of matrices.**

#include <stdio.h>

void inputMatrix(int matrix[][10], int row, int col) {

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

for (int j = 0; j < col; j++) {

printf("Enter element [%d][%d]: ", i, j);

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

}}}

void printMatrix(int matrix[][10], int row, int col) {

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

for (int j = 0; j < col; j++) {

printf("%d\t", matrix[i][j]);

}

printf("\n");}}

void addMatrix(int A[][10], int B[][10], int result[][10], int row, int col) {

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

for (int j = 0; j < col; j++) {

result[i][j] = A[i][j] + B[i][j];}}}

void subtractMatrix(int A[][10], int B[][10], int result[][10], int row, int col) {

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

for (int j = 0; j < col; j++) {

result[i][j] = A[i][j] - B[i][j];}}}

void multiplyMatrix(int A[][10], int B[][10], int result[][10], int row1, int col1, int col2) {

for (int i = 0; i < row1; i++) {

for (int j = 0; j < col2; j++) {

result[i][j] = 0;

for (int k = 0; k < col1; k++) {

result[i][j] += A[i][k] \* B[k][j];}}}}

int main() {

int A[10][10], B[10][10], result[10][10];

int row1, col1, row2, col2;

printf("Enter rows and columns for first matrix: ");

scanf("%d %d", &row1, &col1);

printf("Enter elements for first matrix:\n");

inputMatrix(A, row1, col1);

printf("Enter rows and columns for second matrix: ");

scanf("%d %d", &row2, &col2);

printf("Enter elements for second matrix:\n");

inputMatrix(B, row2, col2);

if (row1 == row2 && col1 == col2) {

addMatrix(A, B, result, row1, col1);

printf("\nAddition of matrices:\n");

printMatrix(result, row1, col1);

} else {

printf("\nAddition not possible! Matrices must have same dimensions.\n");

}

if (row1 == row2 && col1 == col2) {

subtractMatrix(A, B, result, row1, col1);

printf("\nSubtraction of matrices:\n");

printMatrix(result, row1, col1);

} else {

printf("\nSubtraction not possible! Matrices must have same dimensions.\n");

}

if (col1 == row2) {

multiplyMatrix(A, B, result, row1, col1, col2);

printf("\nMultiplication of matrices:\n");

printMatrix(result, row1, col2);

} else {

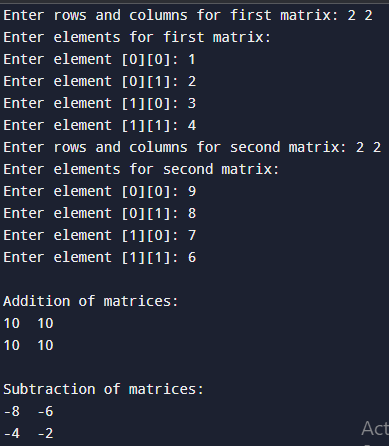
printf("\nMultiplication not possible! Number of columns of first matrix must equal rows of second matrix.\n");

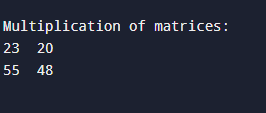
}

return 0;

}

**OUTPUT:**





1. **Write a program to find the largest and the smallest element of a matrix.**

#include <stdio.h>

int main() {

int rows, cols;

printf("Enter number of rows: ");

scanf("%d", &rows);

printf("Enter number of columns: ");

scanf("%d", &cols);

int matrix[rows][cols];

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

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

for (int j = 0; j < cols; j++) {

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

}

}

int largest = matrix[0][0];

int smallest = matrix[0][0];

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

for (int j = 0; j < cols; j++) {

if (matrix[i][j] > largest) {

largest = matrix[i][j];

}

if (matrix[i][j] < smallest) {

smallest = matrix[i][j];

}

}

}

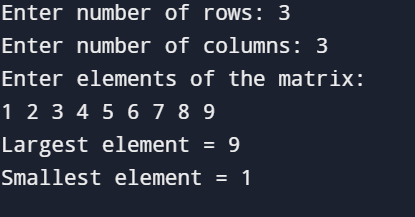
printf("Largest element = %d\n", largest);

printf("Smallest element = %d\n", smallest);

return 0;

}

**OUTPUT:**

****

**4.** **Write a program to calculate the sum of each row and each column and total of all elements of matrix.**

#include <stdio.h>

int main() {

int rows, cols;

printf("Enter number of rows: ");

scanf("%d", &rows);

printf("Enter number of columns: ");

scanf("%d", &cols);

int matrix[rows][cols];

printf(" Enter elements of the %dx%d matrix:\n", rows, cols);

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

printf(" Row %d:\n", i + 1);

for (int j = 0; j < cols; j++) {

printf(" Enter element [%d][%d]: ", i + 1, j + 1);

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

}

}

printf("\n Your Matrix:\n");

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

for (int j = 0; j < cols; j++) {

printf("%4d", matrix[i][j]);

}

printf("\n");

}

int totalSum = 0;

printf("\n Sum of each row:\n");

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

int rowSum = 0;

for (int j = 0; j < cols; j++) {

rowSum += matrix[i][j];

}

printf(" Sum of row %d = %d\n", i + 1, rowSum);

totalSum += rowSum;

}

printf("\nSum of each column:\n");

for (int j = 0; j < cols; j++) {

int colSum = 0;

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

colSum += matrix[i][j];

}

printf(" Sum of column %d = %d\n", j + 1, colSum);

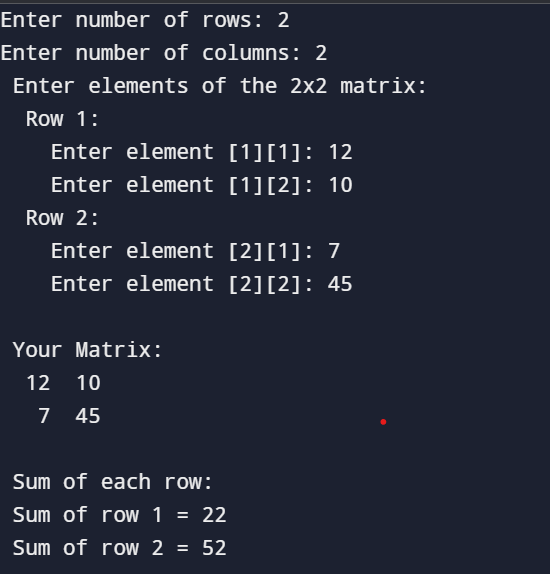
}

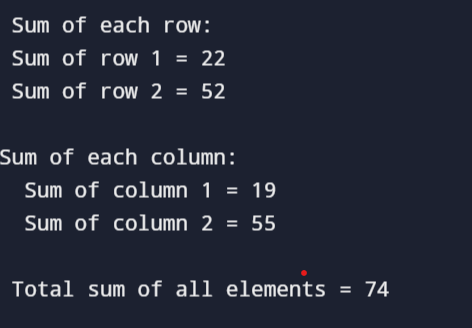
printf("\n Total sum of all elements = %d\n", totalSum);

return 0;

}

**OUTPUT:**

****

****

1. **Write a program to implement linear search.**

#include <stdio.h>

int linear\_search(int arr[], int size, int target) {

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

if (arr[i] == target) {

return i;

}

}

return -1;

}

int main() {

int arr[] = {10, 20, 30, 40, 50};

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

int target = 30;

int result = linear\_search(arr, size, target);

if (result != -1) {

printf("Element found at index %d\n", result);

} else {

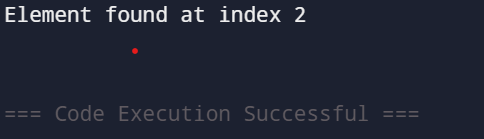
printf("Element not found\n");

}

return 0;

}

**OUTPUT:**

****

1. **Write a program to implement binary search.**

#include <stdio.h>

int binarySearch(int array[], int x, int low, int high) {

while (low <= high) {

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

if (array[mid] == x)

return mid;

if (array[mid] < x)

low = mid + 1;

else

high = mid - 1;

}

return -1;

}

int main(void) {

int array[] = {3, 4, 5, 6, 7, 8, 9};

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

int x = 4;

int result = binarySearch(array, x, 0, n - 1);

if (result == -1)

printf("Not found");

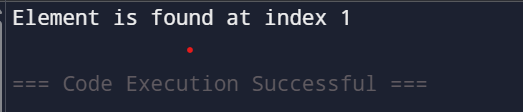
else

printf("Element is found at index %d", result);

return 0;

}:

**OUTPUT:**

****

1. **Write a program to print the sum of digits of the given number.**

#include <stdio.h>

int main() {

int number, sum = 0, digit;

printf("Enter a number: ");

scanf("%d", &number);

int temp = number;

while (temp != 0) {

digit = temp % 10;

sum += digit;

temp /= 10;

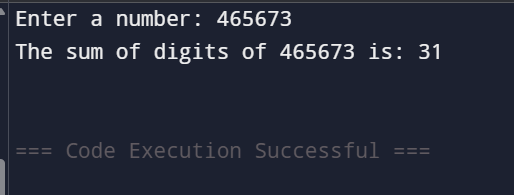
}

printf("The sum of digits of %d is: %d\n", number, sum); vg

return 0;

}

**OUTPUT:**

****

1. **Write a program to check if a number is palindrome or not.**

#include <stdio.h>

int main() {

int number, reversed = 0, remainder, original;

printf("Enter a number: ");

scanf("%d", &number);

original = number;

while (number != 0) {

remainder = number % 10;

reversed = reversed \* 10 + remainder;

number /= 10;

}

if (original == reversed) {

printf("%d is a palindrome!\n", original);

} else {

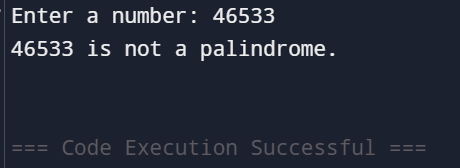
printf("%d is not a palindrome.\n", original);

}

return 0;

}

**OUTPUT:**

****

1. **Write a program to find whether the given number is prime or not.**

#include <stdio.h>

int main() {

int number, i, isPrime = 1;

printf("Enter a number: ");

scanf("%d", &number);

if (number <= 1) {

printf("%d is not a prime number.\n", number);

return 0;

}

for (i = 2; i <= number / 2; i++) {

if (number % i == 0) {

isPrime = 0;

break;

}

}

if (isPrime)

printf("%d is a prime number!\n", number);

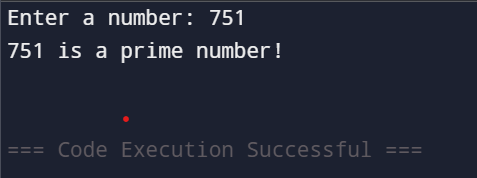
else

printf("%d is not a prime number.\n", number);

return 0;

}

**OUTPUT:**

****

1. **Write a program to print the number of occurrences of respective alphabets A-Z in the given line of text.**

#include <stdio.h>

#include <ctype.h>

int main() {

char text[1000];

int count[26] = {0};

int i;

printf("Enter a line of text: ");

fgets(text, sizeof(text), stdin);

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

char ch = toupper(text[i]);

if (ch >= 'A' && ch <= 'Z') {

count[ch - 'A']++;

}

}

printf("\nOccurrences of each alphabet:\n");

for (i = 0; i < 26; i++) {

if (count[i] > 0) {

printf("%c: %d\n", i + 'A', count[i]);

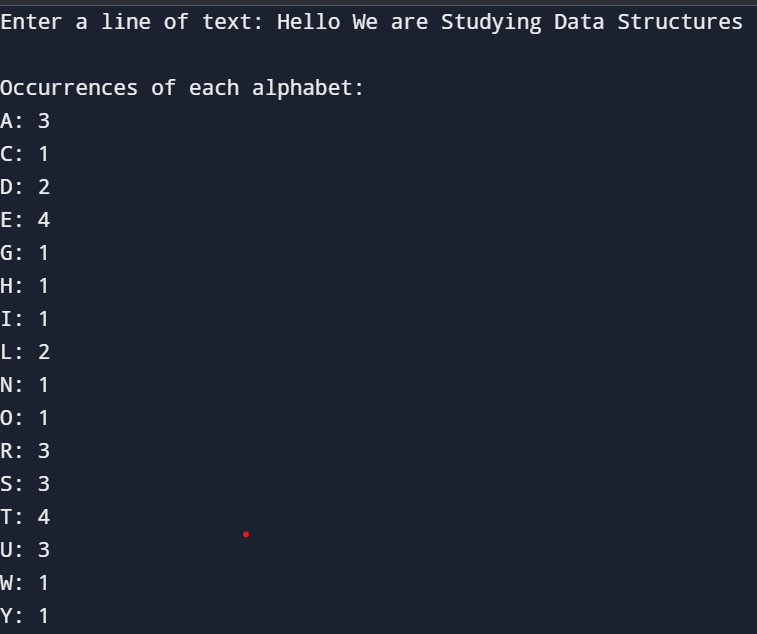
}

}

return 0;

}

**OUTPUT:**

****

**11.Write a program to print the following**

**pattern:**

A B C D E

B C D E A

C D E A B

D E A B C

E A B C D

#include <stdio.h>

int main() {

int total\_letters = 5;

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

for (int row = 0; row < total\_letters; row++) {

for (int col = 0; col < total\_letters; col++) {

int index = (row + col) % total\_letters;

printf("%c ", letters[index]);

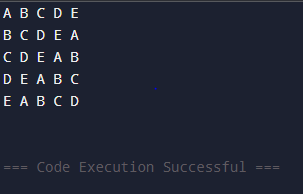
} printf("\n");

}

return 0;

}

**OUTPUT:**



**13.Write a program to implement selection sort.**

#include <stdio.h>

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

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int main() {

int numbers[100];

int size;

printf("Enter the numbers :");

scanf("%d", &size);

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

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

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

}

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

int min\_index = i;

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

if (numbers[j] < numbers[min\_index]) {

min\_index = j;

}

}

if (min\_index != i) {

swap(&numbers[i], &numbers[min\_index]);

}

}

printf("Sorted numbers:\n");

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

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

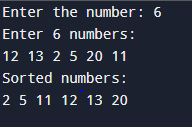
}

printf("\n");

return 0;

}

**OUTPUT:**



**14.Write a program to implement bubble sort.**

#include <stdio.h>

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

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int main() {

int numbers[100];

int size;

printf("How many numbers do you want to sort? ");

scanf("%d", &size);

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

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

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

}

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

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

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

swap(&numbers[j], &numbers[j + 1]);

}

}

}

printf("Sorted numbers:\n");

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

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

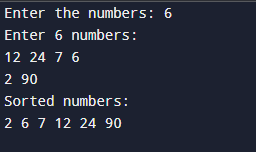
}

printf("\n");

return 0;

}

**OUTPUT:**



**15.Write a program to implement insertion sort.**

#include <stdio.h>

void insertionSort(int arr[], int size) {

// Start from the second element (because the first element is already "sorted")

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

int key = arr[i];

int j = i - 1;

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

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

j = j - 1;

}

arr[j + 1] = key;

}

}

int main() {

int numbers[100];

int size;

printf("Enter the number: ");

scanf("%d", &size);

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

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

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

}

insertionSort(numbers, size);

printf("Sorted numbers:\n");

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

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

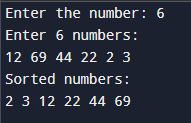
}

printf("\n");

return 0;

}

**OUTPUT:**



**12.** **Write a program to replace any particular word in a line of text by another word.**

#include <stdio.h>

#include <string.h>

#define MAX\_LENGTH 1000

int main() {

char text[MAX\_LENGTH];

char find[100], replace[100];

char result[MAX\_LENGTH];

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

printf("Please enter a line of text:\n");

fgets(text, sizeof(text), stdin);

printf("\nWhich word would you like to replace?\n");

scanf("%s", find);

printf("\nWhat word should it be replaced with?\n");

scanf("%s", replace);

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

found = 0;

if (strncmp(&text[i], find, strlen(find)) == 0 &&

(text[i + strlen(find)] == ' ' || text[i + strlen(find)] == '\n' || text[i + strlen(find)] == '\0')) {

found = 1;

for (k = 0; replace[k] != '\0'; k++, j++) {

result[j] = replace[k];

}

i += strlen(find);

} else {

result[j++] = text[i++];

}

}

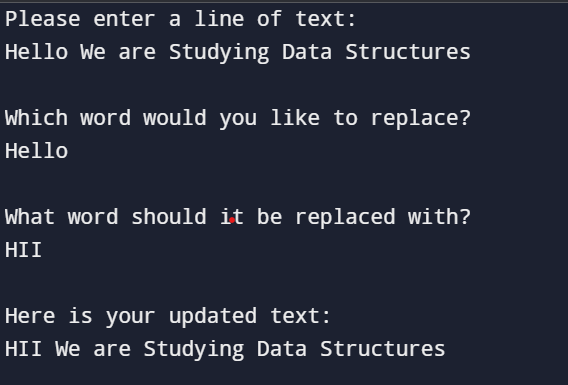
result[j] = '\0'; // Null terminate the result string

printf("\nHere is your updated text:\n%s\n", result);

return 0;

}

**OUTPUT:**

****

**16.Write a program to implement merge sort.**

#include <stdio.h>

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

int i = left;

int j = mid + 1;

int k = 0;

int temp[right - left + 1];

while (i <= mid && j <= right) {

if (arr[i] <= arr[j]) {

temp[k++] = arr[i++];

} else {

temp[k++] = arr[j++];

}

}

while (i <= mid) {

temp[k++] = arr[i++];

}

while (j <= right) {

temp[k++] = arr[j++];

}

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

arr[left + m] = temp[m];

}

}

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);

}

}

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

printf("Sorted array: ");

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

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

}

printf("\n");

}

int main() {

int arr[] = {38, 27, 43, 3, 9, 82, 10};

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

printf("Original array: ");

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

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

}

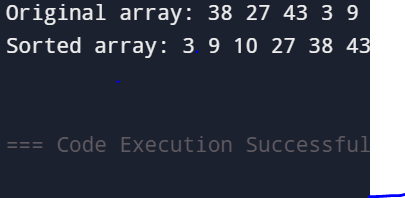
printf("\n");

mergeSort(arr, 0, size - 1);

printArray(arr, size);

return 0;

}**OUTPUT:**



**17.Write a program to convert a matrix into lower triangular sparse matrix.**

#include <stdio.h>

#define MAX 10

void printMatrix(int matrix[MAX][MAX], int rows, int cols) {

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

for (int j = 0; j < cols; j++) {

printf("%d ", matrix[i][j]);

}

printf("\n");

}

}

int main() {

int matrix[MAX][MAX];

int lower[MAX][MAX];

int rows, cols;

printf("Enter the number of rows and columns of the matrix: ");

scanf("%d %d", &rows, &cols);

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

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

for (int j = 0; j < cols; j++) {

printf("Element [%d][%d]: ", i, j);

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

}

}

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

for (int j = 0; j < cols; j++) {

if (i >= j) {

lower[i][j] = matrix[i][j];

} else {

lower[i][j] = 0; }

}

}

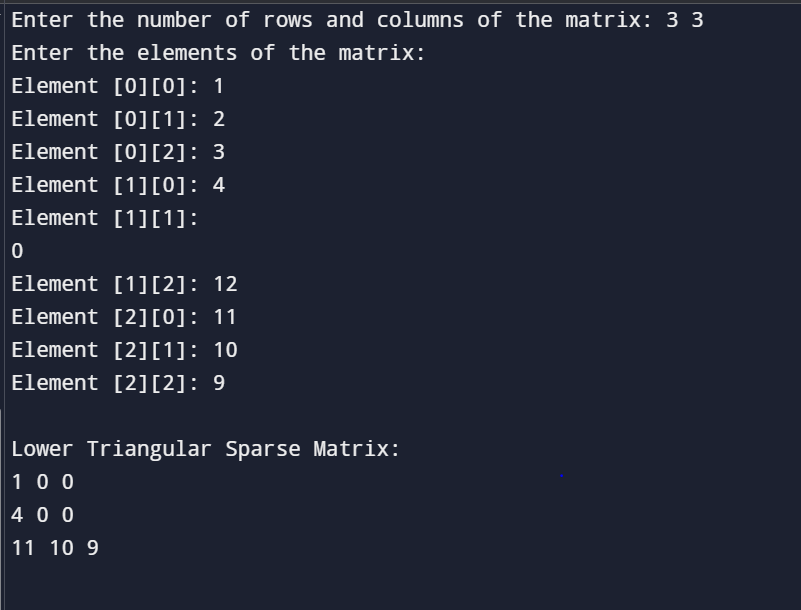
printf("\nLower Triangular Sparse Matrix:\n");

printMatrix(lower, rows, cols);

return 0;

}

**OUTPUT:**



**18.Write a program to convert a matrix into upper triangular sparse matrix.**

#include <stdio.h>

#define MAX 10

void printMatrix(int matrix[MAX][MAX], int rows, int cols) {

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

for (int j = 0; j < cols; j++) {

printf("%d ", matrix[i][j]);

}

printf("\n");

}

}

int main() {

int matrix[MAX][MAX];

int upper[MAX][MAX];

int rows, cols;

printf("Enter number of rows and columns: ");

scanf("%d %d", &rows, &cols);

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

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

for (int j = 0; j < cols; j++) {

printf("Element [%d][%d]: ", i, j);

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

}

}

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

for (int j = 0; j < cols; j++) {

if (i <= j) {

upper[i][j] = matrix[i][j];

} else {

upper[i][j] = 0;

}

}

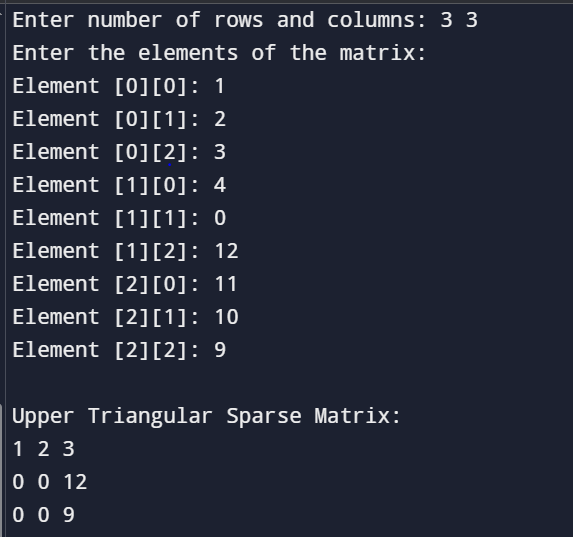
}

printf("\nUpper Triangular Sparse Matrix:\n");

printMatrix(upper, rows, cols);

return 0;}

**OUTPUT:**



**19.Write a program to check whether the given matrix is lower triangular sparse matrix or not.**

#include <stdio.h>

#define MAX 10

void inputMatrix(int matrix[MAX][MAX], int rows, int cols) {

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

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

for (int j = 0; j < cols; j++) {

printf("Element [%d][%d]: ", i, j);

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

}

}

}

void printMatrix(int matrix[MAX][MAX], int rows, int cols) {

printf("\nMatrix:\n");

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

for (int j = 0; j < cols; j++) {

printf("%d ", matrix[i][j]);

}

printf("\n");

}

}

int isLowerTriangularSparse(int matrix[MAX][MAX], int rows, int cols) {

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

for (int j = 0; j < cols; j++) {

if (i < j && matrix[i][j] != 0) {

return 0;

}

}

}

return 1;

}

int main() {

int matrix[MAX][MAX];

int rows, cols;

printf("Enter number of rows and columns: ");

scanf("%d %d", &rows, &cols);

inputMatrix(matrix, rows, cols);

printMatrix(matrix, rows, cols);

if (isLowerTriangularSparse(matrix, rows, cols)) {

printf("\nYes! The matrix is a Lower Triangular Sparse Matrix ✅\n");

} else {

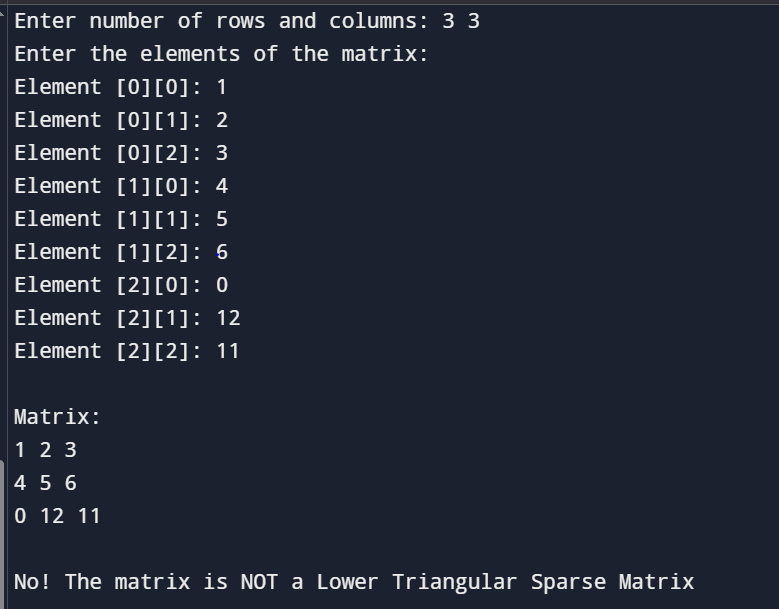
printf("\nNo! The matrix is NOT a Lower Triangular Sparse Matrix ❌\n");

}

return 0;

}

**OUTPUT:**



**20.Write a program to check whether the given matrix is upper triangular sparse matrix or not.**

#include <stdio.h>

#define MAX 10

void inputMatrix(int matrix[MAX][MAX], int rows, int cols) {

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

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

for (int j = 0; j < cols; j++) {

printf("Element [%d][%d]: ", i, j);

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

}

}

}

void printMatrix(int matrix[MAX][MAX], int rows, int cols) {

printf("\nMatrix:\n");

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

for (int j = 0; j < cols; j++) {

printf("%d ", matrix[i][j]);

}

printf("\n");

}

}

int isUpperTriangularSparse(int matrix[MAX][MAX], int rows, int cols) {

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

for (int j = 0; j < cols; j++) {

if (i > j && matrix[i][j] != 0) {

return 0;

}

}

}

return 1;

}

int main() {

int matrix[MAX][MAX];

int rows, cols;

printf("Enter number of rows and columns: ");

scanf("%d %d", &rows, &cols);

inputMatrix(matrix, rows, cols);

printMatrix(matrix, rows, cols);

if (isUpperTriangularSparse(matrix, rows, cols)) {

printf("\nYes! The matrix is an Upper Triangular Sparse Matrix ✅\n");

} else {

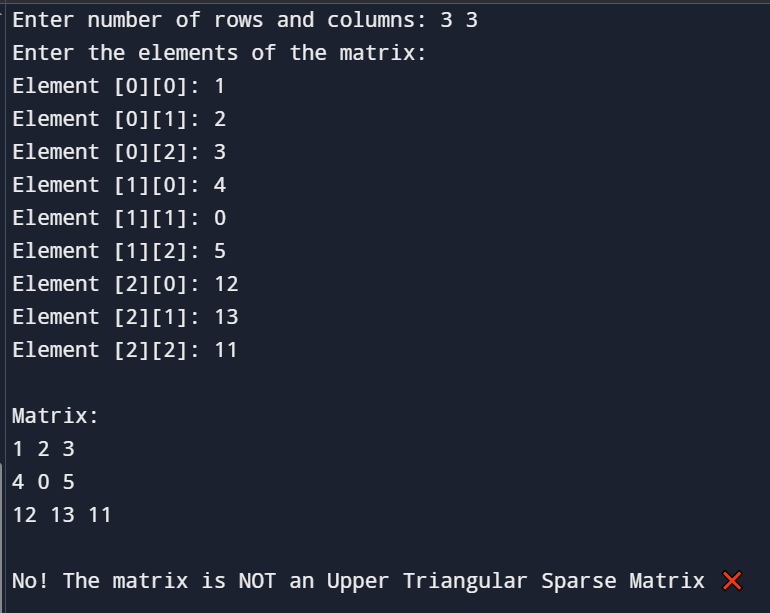
printf("\nNo! The matrix is NOT an Upper Triangular Sparse Matrix ❌\n");

}

return 0;

}

**OUTPUT:**



**21.Write a program to convert a matrix into tridiagonal matrix.**

#include <stdio.h>

#define MAX 10

void inputMatrix(int matrix[MAX][MAX], int size) {

printf("Enter the elements of the %dx%d matrix:\n", size, size);

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

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

printf("Element [%d][%d]: ", i, j);

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

}

}

}

void printMatrix(int matrix[MAX][MAX], int size) {

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

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

printf("%d ", matrix[i][j]);

}

printf("\n");

}

}

void convertToTridiagonal(int matrix[MAX][MAX], int size, int result[MAX][MAX]) {

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

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

// Keep only the main, upper, and lower diagonals

if (i == j || i == j + 1 || i == j - 1) {

result[i][j] = matrix[i][j];

} else {

result[i][j] = 0;

}

}

}

}

int main() {

int matrix[MAX][MAX], tridiagonal[MAX][MAX];

int size;

printf("Enter the size of the square matrix: ");

scanf("%d", &size);

inputMatrix(matrix, size);

convertToTridiagonal(matrix, size, tridiagonal);

printf("\nOriginal Matrix:\n");

printMatrix(matrix, size);

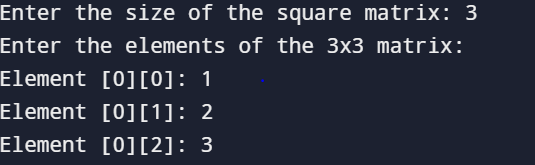
printf("\nTridiagonal Matrix:\n");

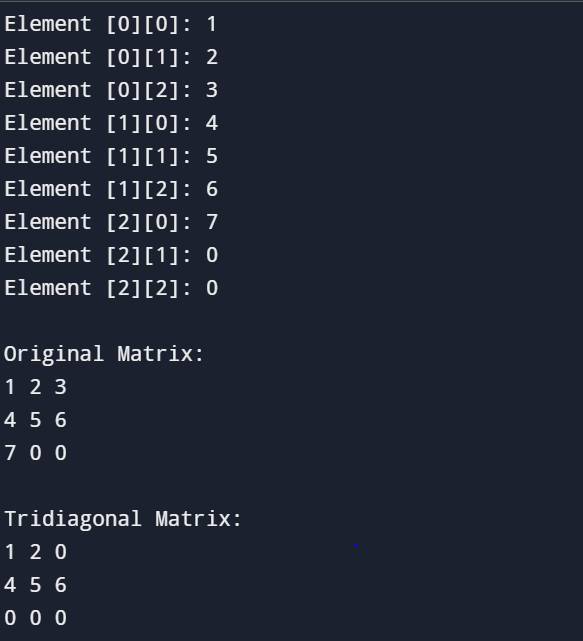
printMatrix(tridiagonal, size);

return 0;

}

**OUTPUT:**





**22.Write a program to show a sparse matrix in tuple form.**

#include <stdio.h>

int main() {

int matrix[10][10];

int rows, cols;

int i, j;

int tuple[100][3];

int k = 1;

printf("Enter number of rows and columns: ");

scanf("%d %d", &rows, &cols);

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

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

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

printf("Element at [%d][%d]: ", i, j);

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

}

}

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

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

if(matrix[i][j] != 0) {

tuple[k][0] = i;

tuple[k][1] = j;

tuple[k][2] = matrix[i][j];

k++;

}

}

}

tuple[0][0] = rows;

tuple[0][1] = cols;

tuple[0][2] = k - 1;

printf("\nSparse Matrix in Tuple Form:\n");

printf("Row Col Value\n");

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

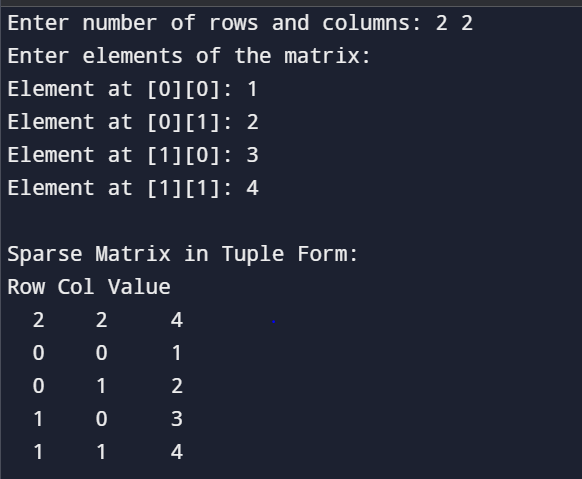
printf("%3d %4d %5d\n", tuple[i][0], tuple[i][1], tuple[i][2]);

}

return 0;

}

**OUTPUT:**



**23.Suppose X, Y, Z are integer arrays of m, n and m+n sizes respectively. The number in array X appears in ascending order and the numbers in array Y appears in descending order. Give an algorithm to produce a third array Z, containing all data of array X and Y in ascending order.**

#include <stdio.h>

void mergeArrays(int X[], int Y[], int Z[], int m, int n) {

int i = 0;

int j = n - 1;

int k = 0;

while (i < m && j >= 0) {

if (X[i] < Y[j]) {

Z[k++] = X[i++];

} else {

Z[k++] = Y[j--];

}

}

while (i < m) {

Z[k++] = X[i++];

}

while (j >= 0) {

Z[k++] = Y[j--];

}

}

int main() {

int m, n;

printf("Enter size of array X (ascending): ");

scanf("%d", &m);

printf("Enter size of array Y (descending): ");

scanf("%d", &n);

int X[m], Y[n], Z[m + n];

printf("Enter %d elements for array X (ascending order):\n", m);

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

printf("X[%d]: ", i);

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

}

printf("Enter %d elements for array Y (descending order):\n", n);

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

printf("Y[%d]: ", i);

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

}

mergeArrays(X, Y, Z, m, n);

printf("\nMerged array Z in ascending order:\n");

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

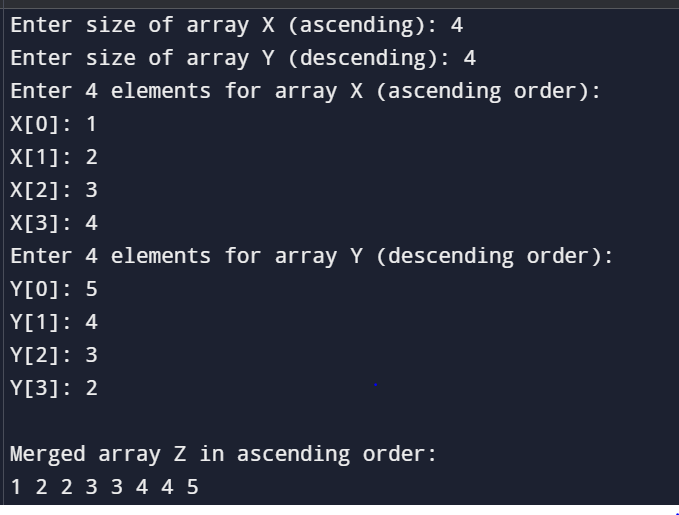
printf("%d ", Z[i]);

}

printf("\n");

return 0;

}

**OUTPUT:** 

**24.Write a program of single linked list and perform the operations on it like (insert, display, length, delete).**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* head = NULL;

void insertNode(int value) {

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

newNode->data = value;

newNode->next = NULL;

if (head == NULL) {

head = newNode;

} else {

struct Node\* temp = head;

while (temp->next != NULL)

temp = temp->next;

temp->next = newNode;

}

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

}

void displayList() {

if (head == NULL) {

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

return;

}

struct Node\* temp = head;

printf("Linked List: ");

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

int getLength() {

int count = 0;

struct Node\* temp = head;

while (temp != NULL) {

count++;

temp = temp->next;

}

return count;

}

void deleteNode(int value) {

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

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

head = temp->next;

free(temp);

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

return;

}

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

prev = temp;

temp = temp->next;

}

if (temp == NULL) {

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

return;

}

prev->next = temp->next;

free(temp);

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

}

int main() {

int choice, value;

while (1) {

printf("1. Insert Node\n");

printf("2. Display List\n");

printf("3. Length of List\n");

printf("4. Delete Node\n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to insert: ");

scanf("%d", &value);

insertNode(value);

break;

case 2:

displayList();

break;

case 3:

printf("Length of list: %d\n", getLength());

break;

case 4:

printf("Enter value to delete: ");

scanf("%d", &value);

deleteNode(value);

break;

case 5:

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

exit(0);

default:

printf("Invalid choice. Try again.\n");

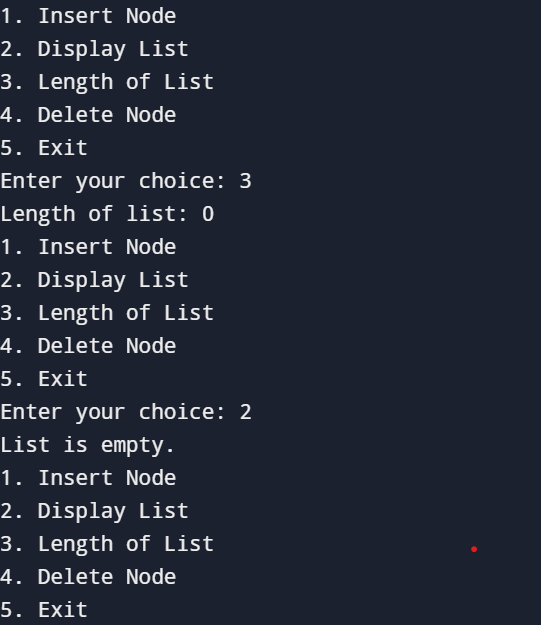
}

}

return 0;

}

**OUTPUT:**

****

**25.** **Write a program of Double linked list and perform the operations on it like (Insert, display, length, delete).**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* prev;

struct Node\* next;

};

struct Node\* head = NULL;

struct Node\* createNode(int value) {

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

newNode->data = value;

newNode->prev = NULL;

newNode->next = NULL;

return newNode;

}

void insert(int value) {

struct Node\* newNode = createNode(value);

if (head == NULL) {

head = newNode;

printf("%d inserted as the first node.\n", value);

} else {

struct Node\* temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

newNode->prev = temp;

printf("%d inserted at the end.\n", value);

}

}

void display() {

if (head == NULL) {

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

return;

}

struct Node\* temp = head;

printf("Doubly Linked List: ");

while (temp != NULL) {

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

temp = temp->next;

}

printf("\n");

}

int length() {

int count = 0;

struct Node\* temp = head;

while (temp != NULL) {

count++;

temp = temp->next;

}

return count;

}

void deleteNode(int value) {

if (head == NULL) {

printf("List is empty. Nothing to delete.\n");

return;

}

struct Node\* temp = head;

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

temp = temp->next;

}

if (temp == NULL) {

printf("Node with value %d not found.\n", value);

return;

}

if (temp == head) {

head = temp->next;

if (head != NULL) {

head->prev = NULL;

}

} else {

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

if (temp->next != NULL) {

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

}

}

free(temp);

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

}

int main() {

int choice, value;

printf(" Doubly Linked List Operations \n");

while (1) {

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

printf("1. Insert\n");

printf("2. Display\n");

printf("3. Length\n");

printf("4. Delete\n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to insert: ");

scanf("%d", &value);

insert(value);

break;

case 2:

display();

break;

case 3:

printf("Length of the list: %d\n", length());

break;

case 4:

printf("Enter value to delete: ");

scanf("%d", &value);

deleteNode(value);

break;

case 5:

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

exit(0);

default:

printf("Invalid choice. Please select between 1-5.\n");

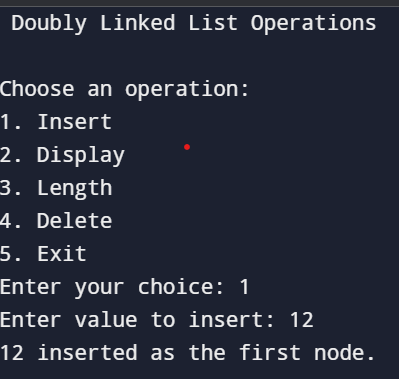
}

}

return 0;

}

**OUTPUT:**

****

**26.Write a program of circular single linked list and perform the operations on it like (insert, display, length, delete).**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* head = NULL;

struct Node\* createNode(int value) {

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

newNode->data = value;

newNode->next = NULL;

return newNode;

}

void insert(int value) {

struct Node\* newNode = createNode(value);

if (head == NULL) {

head = newNode;

head->next = head; // Circular link

printf("%d is inserted as the first node.\n", value);

} else {

struct Node\* temp = head;

while (temp->next != head) {

temp = temp->next;

}

temp->next = newNode;

newNode->next = head;

printf("%d is inserted at the end.\n", value);

}

}

void display() {

if (head == NULL) {

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

return;

}

struct Node\* temp = head;

printf("Here’s your Circular Linked List: ");

do {

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

temp = temp->next;

} while (temp != head);

printf("\n");

}

int length() {

if (head == NULL)

return 0;

int count = 0;

struct Node\* temp = head;

do {

count++;

temp = temp->next;

} while (temp != head);

return count;

}

void deleteNode(int value) {

if (head == NULL) {

printf("The list is empty. Nothing to delete!\n");

return;

}

struct Node\* current = head;

struct Node\* previous = NULL;

if (head->data == value) {

if (head->next == head) {

free(head);

head = NULL;

printf("%d was the only node and now list is empty.\n", value);

return;

}

struct Node\* last = head;

while (last->next != head) {

last = last->next;

}

last->next = head->next;

struct Node\* temp = head;

head = head->next;

free(temp);

printf("Head node with value %d deleted.\n", value);

return;

}

do {

previous = current;

current = current->next;

} while (current != head && current->data != value);

if (current->data == value) {

previous->next = current->next;

free(current);

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

} else {

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

}

}

int main() {

int choice, value;

printf("\n Welcome to Circular Singly Linked List World\n");

while (1) {

printf("\nWhat do you want to do?\n");

printf("1. Insert a node\n");

printf("2. Display the list\n");

printf("3. Check the length of list\n");

printf("4. Delete a node\n");

printf("5. Exit the program\n");

printf("Enter your choice (1-5): ");

scanf("%d", &choice);

switch (choice) {

case 1:

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

scanf("%d", &value);

insert(value);

break;

case 2:

display();

break;

case 3:

printf("Length of the list is: %d\n", length());

break;

case 4:

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

scanf("%d", &value);

deleteNode(value);

break;

case 5:

printf("Thank you for using the program. Bye Bye!\n");

exit(0);

default:

printf("Invalid choice! Please choose between 1 to 5.\n");

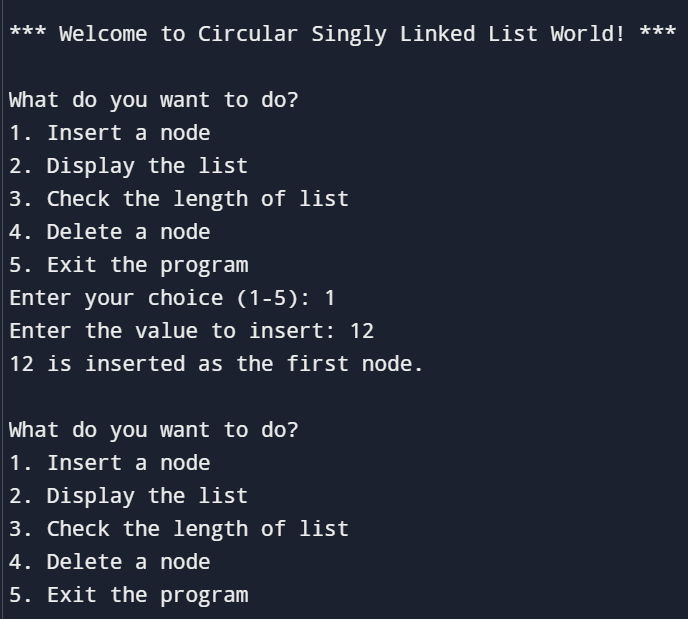
}

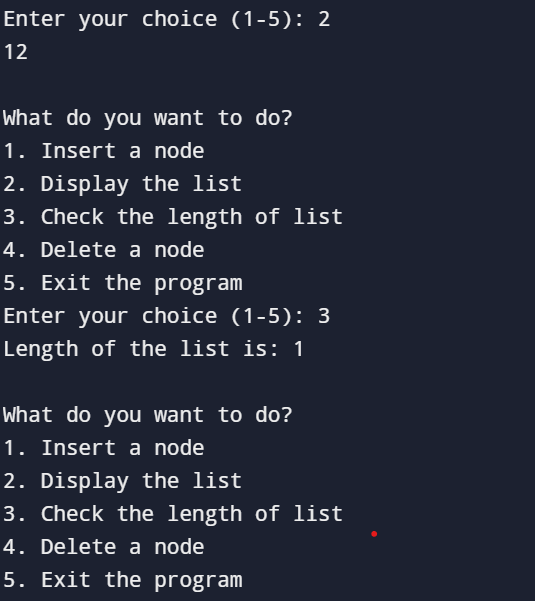
}

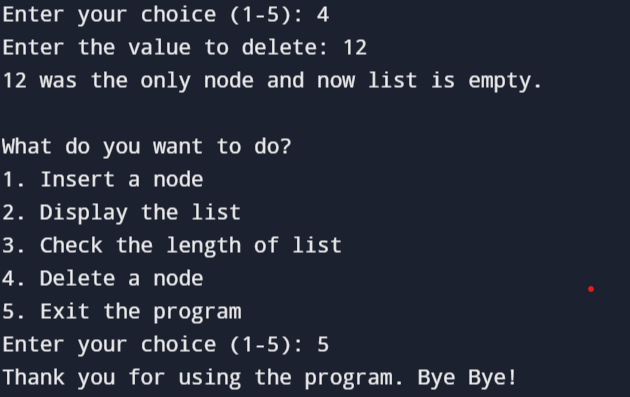
return 0;

}

**OUTPUT:**

****

****

****

**27.** **Write a program of circular Double linked list and perform the operations on it like (insert, display, length, delete).**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

struct Node\* prev; };

struct Node\* head = NULL;

struct Node\* createNode(int value) {

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

newNode->data = value;

newNode->next = NULL;

newNode->prev = NULL;

return newNode;

}

void insert(int value) {

struct Node\* newNode = createNode(value);

if (head == NULL) {

head = newNode;

head->next = head;

head->prev = head;

printf("%d inserted as the first node.\n", value);

} else {

struct Node\* last = head->prev;

last->next = newNode;

newNode->prev = last;

newNode->next = head;

head->prev = newNode;

printf("%d inserted at the end.\n", value);

}

}

void display() {

if (head == NULL) {

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

return;

}

struct Node\* temp = head;

printf("Circular Doubly Linked List: ");

do {

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

temp = temp->next;

} while (temp != head);

printf("\n");

}

int length() {

if (head == NULL)

return 0;

int count = 0;

struct Node\* temp = head;

do {

count++;

temp = temp->next;

} while (temp != head);

return count;

}

void deleteNode(int value) {

if (head == NULL) {

printf("The list is empty. Nothing to delete!\n");

return;

}

struct Node\* current = head;

do {

if (current->data == value) {

if (current->next == current) {

free(current);

head = NULL;

printf("Deleted the only node with value %d. List is now empty.\n", value);

return;

}

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

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

if (current == head) {

head = current->next;

}

free(current);

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

return;

}

current = current->next;

} while (current != head);

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

}

int main() {

int choice, value;

printf("\nWelcome to Circular Doubly Linked List World\n");

while (1) {

printf("\nWhat would you like to do?\n");

printf("1. Insert a node\n");

printf("2. Display the list\n");

printf("3. Check the length of list\n");

printf("4. Delete a node\n");

printf("5. Exit the program\n");

printf("Enter your choice (1-5): ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the value you want to insert: ");

scanf("%d", &value);

insert(value);

break;

case 2:

display();

break;

case 3:

printf("Length of the list: %d\n", length());

break;

case 4:

printf("Enter the value you want to delete: ");

scanf("%d", &value);

deleteNode(value);

break;

case 5:

printf("Thanks for using the program! Goodbye! 👋\n");

exit(0);

default:

printf("Invalid choice! Please choose between 1 and 5.\n");

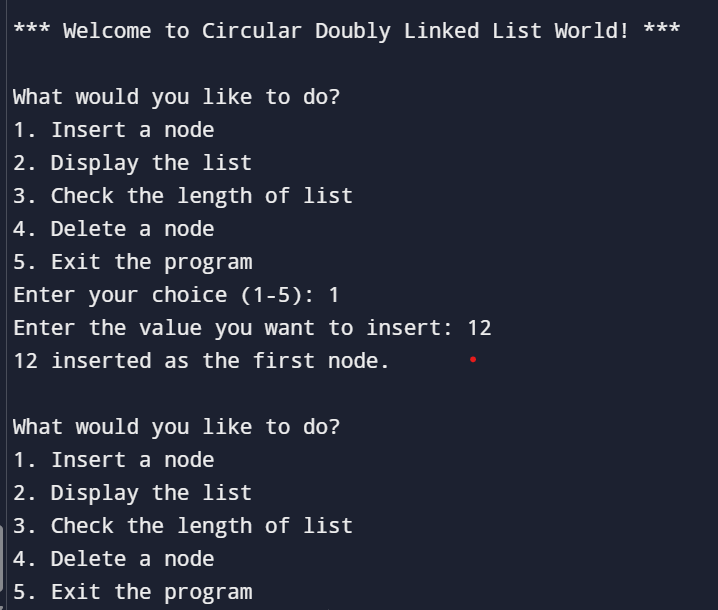
}

}

return 0;

}

**OUTPUT:**

****

**28.** **Write a program of Static Implementation of stack.**

#include <stdio.h>

#define MAX 5

int stack[MAX];

int top = -1;

void push(int value) {

if (top == MAX - 1) {

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

} else {

top++;

stack[top] = value;

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

}

}

void pop() {

if (top == -1) {

printf("Stack Underflow Cannot pop\n");

} else {

printf("%d popped from stack\n", stack[top]);

top--;

}

}

void display() {

if (top == -1) {

printf("Stack is empty\n");

} else {

printf("Stack elements are:\n");

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

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

}

}

}

int main() {

int choice, value;

while (1) {

printf("\n--- Stack Menu ---\n");

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

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to push: ");

scanf("%d", &value);

push(value);

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

printf("Exiting program.\n");

return 0;

default:

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

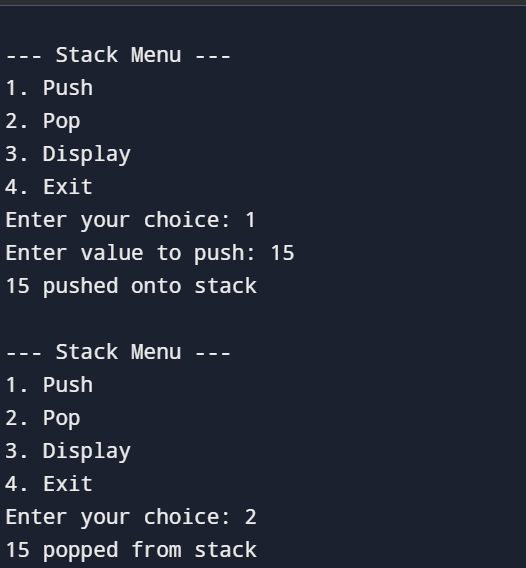
}

}

return 0;

}

**OUTPUT:**

****

**29. Write a program of Dynamic Implementation of stack.**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* top = NULL;

void push(int value) {

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

if (newNode == NULL) {

printf("Memory allocation failed! Cannot push %d\n", value);

return;

}

newNode->data = value;

newNode->next = top;

top = newNode;

printf("%d pushed onto stack!\n", value);

}

void pop() {

if (top == NULL) {

printf("Stack Underflow! No element to pop.\n");

return;

}

struct Node\* temp = top;

printf("%d popped from stack!\n", temp->data);

top = top->next;

free(temp);

}

void display() {

if (top == NULL) {

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

return;

}

printf("Stack elements are:\n");

struct Node\* temp = top;

while (temp != NULL) {

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

temp = temp->next;

}

}

int main() {

int choice, value;

printf("Welcome to Dynamic Stack Implementation in C!\n");

while (1) {

printf("\n--- Stack Menu ---\n");

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

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to push: ");

scanf("%d", &value);

push(value);

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

printf("Thank you\n");

exit(0);

default:

printf("Invalid choice, please try again!\n");

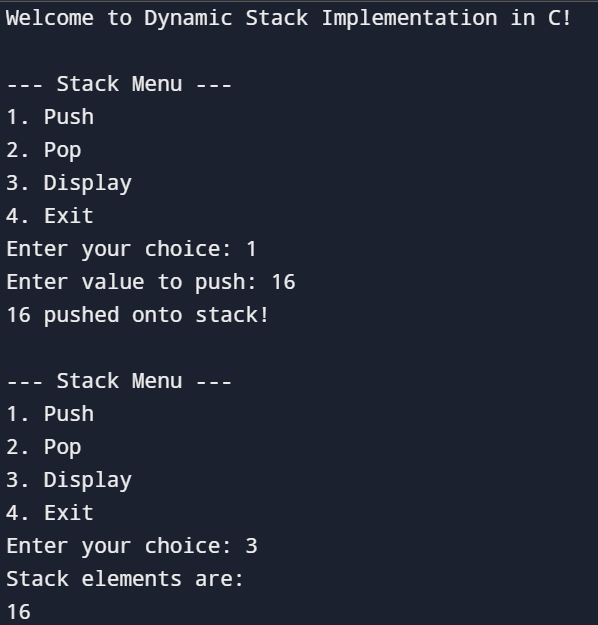
}

}

return 0;

}

**OUTPUT:**

****

**30.** **Write a program to find the minimum and maximum element of the stack.**

#include <stdio.h>

#define SIZE 100

struct Stack {

int numbers[SIZE];

int top;

};

void start(struct Stack \*s) {

s->top = -1;

}

int empty(struct Stack \*s) {

return s->top == -1;

}

int full(struct Stack \*s) {

return s->top == SIZE - 1;

}

void put(struct Stack \*s, int num) {

if (full(s)) {

printf("Stack is full! Cannot add %d\n", num);

} else {

s->numbers[++s->top] = num;

}

}

void findSmallBig(struct Stack \*s) {

if (empty(s)) {

printf("Stack is empty! Nothing to find.\n");

return;

}

int small = s->numbers[0];

int big = s->numbers[0];

for (int i = 1; i <= s->top; i++) {

if (s->numbers[i] < small) {

small = s->numbers[i];

}

if (s->numbers[i] > big) {

big = s->numbers[i];

}

}

printf("Smallest number in stack: %d\n", small);

printf("Biggest number in stack: %d\n", big);

}

int main() {

struct Stack myStack;

start(&myStack);

int howMany, number;

printf("How many numbers you want to add? ");

scanf("%d", &howMany);

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

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

scanf("%d", &number);

put(&myStack, number);

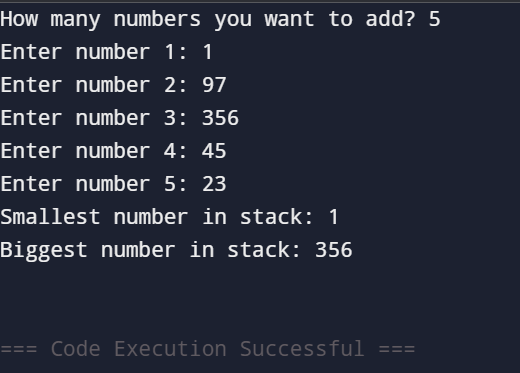
}

findSmallBig(&myStack);

return 0;

}

**OUTPUT:**

****

## 31. Write a program of Static Implementation of linear queue.

#include <stdio.h> #include <stdlib.h> #define MAX 100

struct Queue { int arr[MAX]; int front;

int rear;

};

void initQueue(struct Queue\* queue) { queue->front = -1;

queue->rear = -1;

}

int isFull(struct Queue\* queue) {

return (queue->rear + 1) % MAX == queue->front;

}

int isEmpty(struct Queue\* queue)

{ return queue->front == -1;

}

void enqueue(struct Queue\* queue, int value) { if (isFull(queue)) {

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

if

(isEmpty(queue) while (1)

printf("%d ", queue->arr[i]); if (i == queue->rear) break; i = (i + 1) % MAX;

}

printf("\n");

}

int main() {

struct Queue queue; initQueue(&queue); enqueue(&queue, 10);

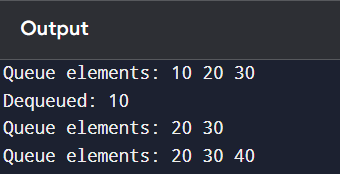
enqueue(&queue, 20);

enqueue(&queue, 30); display(&queue);

printf("Dequeued: %d\n", dequeue(&queue)); display(&queue);

enqueue(&queue, 40); display(&queue); return 0;

}



**32. Write a program of Dynamic Implementation of linear queue.**

#include <stdio.h>

#include <stdlib.h>

#define SIZE 100

typedef struct {

int \*items;

int front, rear;

} Queue;

void initQueue(Queue \*q) {

q->items = (int \*)malloc(SIZE \* sizeof(int));

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

}

int isEmpty(Queue \*q) {

return q->front == -1;

}

int isFull(Queue \*q) {

return q->rear == SIZE - 1;

}

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

if (isFull(q)) {

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

return;

}

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

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

printf("%d enqueued.\n", value);

}

int dequeue(Queue \*q) {

if (isEmpty(q)) {

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

return -1; }

int value = q->items[q->front++];

if (q->front > q->rear) // Reset when queue becomes empty

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

return value;

}

void display(Queue \*q) {

if (isEmpty(q)) {

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

return;}

printf("Queue: ");

for (int i = q->front; i <= q->rear; i++)

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

printf("\n");}

int main() {

Queue q;

initQueue(&q);

enqueue(&q, 10);

enqueue(&q, 20);

enqueue(&q, 30);

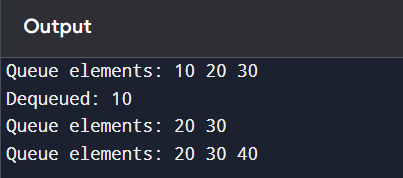
display(&q);

printf("Dequeued: %d\n", dequeue(&q));

display(&q);

free(q.items);

return 0;}

**OUTPUT:**

**33. Write a program of Static Implementation of circular queue**

#include <stdio.h>

#define SIZE 5

typedef struct {

int items[SIZE];

int front, rear;} Queue;

void initQueue(Queue \*q) {

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

int isFull(Queue \*q) {

return (q->rear + 1) % SIZE == q->front;}

int isEmpty(Queue \*q) {

return q->front == -1;}

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

if (isFull(q)) {

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

return; }

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

q->rear = (q->rear + 1) % SIZE;

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

printf("%d enqueued.\n", value);}

int dequeue(Queue \*q) {

if (isEmpty(q)) {

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

return -1; }

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

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

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

else

q->front = (q->front + 1) % SIZE;

return value;}

void display(Queue \*q) {

if (isEmpty(q)) {

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

return;}

printf("Queue: ");

int i = q->front;

while (1) {

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

if (i == q->rear) break;

i = (i + 1) % SIZE;}

printf("\n");}

int main() {

Queue q;

initQueue(&q);

enqueue(&q, 10);

enqueue(&q, 20);

enqueue(&q, 30);

enqueue(&q, 40);

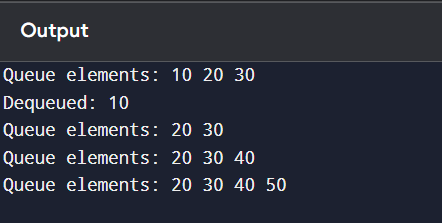
display(&q);

printf("Dequeued: %d\n", dequeue(&q));

enqueue(&q, 50);

display(&q);

return 0;}

**OUTPUT:**

**34. Write a program of Dynamic Implementation of circular queue.**

#include <stdio.h>

#include <stdlib.h>

typedef struct {

int \*items;

int size;

int front, rear;} Queue;

void initQueue(Queue \*q, int size) {

q->size = size;

q->items = (int \*)malloc(size \* sizeof(int));

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

int isFull(Queue \*q) {

return (q->rear + 1) % q->size == q->front;}

int isEmpty(Queue \*q) {

return q->front == -1;}

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

if (isFull(q)) {

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

return;}

if (isEmpty(q))

q->front = 0;

q->rear = (q->rear + 1) % q->size;

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

printf("%d enqueued.\n", value);}

int dequeue(Queue \*q) {

if (isEmpty(q)) {

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

return -1; }

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

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

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

else

q->front = (q->front + 1) % q->size;

return value;}

void display(Queue \*q) {

if (isEmpty(q)) {

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

return; }

printf("Queue: ");

int i = q->front;

while (1) {

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

if (i == q->rear) break;

i = (i + 1) % q->size; }

printf("\n");}

int main() {

Queue q;

int n = 5;

initQueue(&q, n);

enqueue(&q, 10);

enqueue(&q, 20);

enqueue(&q, 30);

enqueue(&q, 40);

display(&q);

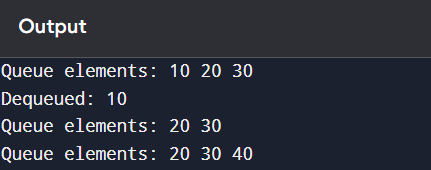
printf("Dequeued: %d\n", dequeue(&q));

enqueue(&q, 50);

display(&q);

free(q.items);

return 0;}

**OUTPUT:**

**35.** **Write a program of Static implementation of double ended queue.**

#include <stdio.h>

#define SIZE 5

typedef struct {

int arr[SIZE];

int front, rear;} Deque;

void initDeque(Deque \*dq) {

dq->front = -1;

dq->rear = -1;}

int isFull(Deque \*dq) {

return (dq->front == 0 && dq->rear == SIZE - 1) || (dq->front == dq->rear + 1);}

int isEmpty(Deque \*dq) {

return dq->front == -1;}

void insertFront(Deque \*dq, int value) {

if (isFull(dq)) {

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

return; }

if (isEmpty(dq)) {

dq->front = dq->rear = 0;

} else if (dq->front == 0) {

dq->front = SIZE - 1;

} else {

dq->front--;}

dq->arr[dq->front] = value;

printf("%d inserted at front.\n", value);}

void insertRear(Deque \*dq, int value) {

if (isFull(dq)) {

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

return;}

if (isEmpty(dq)) {

dq->front = dq->rear = 0;

} else if (dq->rear == SIZE - 1) {

dq->rear = 0;

} else {

dq->rear++;}

dq->arr[dq->rear] = value;

printf("%d inserted at rear.\n", value);}

int deleteFront(Deque \*dq) {

if (isEmpty(dq)) {

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

return -1; }

int value = dq->arr[dq->front];

if (dq->front == dq->rear) {

dq->front = dq->rear = -1;

} else if (dq->front == SIZE - 1) {

dq->front = 0;

} else {

dq->front++; }

return value;}

int deleteRear(Deque \*dq) {

if (isEmpty(dq)) {

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

return -1;}

int value = dq->arr[dq->rear];

if (dq->front == dq->rear) {

dq->front = dq->rear = -1;

} else if (dq->rear == 0) {

dq->rear = SIZE - 1;

} else {

dq->rear--; }

return value;}

void display(Deque \*dq) {

if (isEmpty(dq)) {

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

return;}

printf("Deque: ");

int i = dq->front;

while (1) {

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

if (i == dq->rear) break;

i = (i + 1) % SIZE;}

printf("\n");}

int main() {

Deque dq;

initDeque(&dq);

insertRear(&dq, 10);

insertRear(&dq, 20);

insertFront(&dq, 5);

display(&dq);

printf("Deleted from rear: %d\n", deleteRear(&dq));

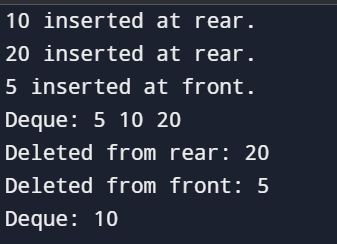
printf("Deleted from front: %d\n", deleteFront(&dq));

display(&dq);

return 0;

}

**OUTPUT:**

****

**36. Write a program of Dynamic implementation of double ended queue.**

#include <stdio.h> #include <stdlib.h> struct Node { int data;

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

};

struct Deque {

struct Node\* front; struct Node\* rear;

};

void initDeque(struct Deque\* deque) { deque->front = NULL; deque->rear = NULL;

}

int isEmpty(struct Deque\* deque) { return deque->front == NULL;

}

void insertFront(struct Deque\* deque, int value) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node)); if (newNode == NULL) { printf("Memory allocation failed. Cannot insert %d at front\n", value); return;

}

newNode->data = value; newNode->next = deque->front; newNode->prev = NULL; if (isEmpty(deque)) {

deque->rear = newNode;

} else {

deque->front->prev = newNode;

}

deque->front = newNode;

}

void insertRear(struct Deque\* deque, int value) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node)); if (newNode == NULL) {

printf("Memory allocation failed. Cannot insert %d at rear\n", value); return;

}

newNode->data = value; newNode->next = NULL; newNode->prev = deque->rear; if (isEmpty(deque)) {

deque->front = newNode;

} else {

deque->rear->next = newNode;

}

deque->rear = newNode;

}

int deleteFront(struct Deque\* deque) { if (isEmpty(deque)) { printf("Deque underflow. Cannot delete from front\n"); return -1;

}

struct Node\* temp = deque->front; int deletedValue = temp->data; deque->front = deque->front->next; if (deque->front != NULL) { deque->front->prev = NULL;

} else {

deque->rear = NULL;

}

free(temp);

return deletedValue;

}

int deleteRear(struct Deque\* deque) { if (isEmpty(deque)) { printf("Deque underflow. Cannot delete from rear\n"); return -1; //

}

struct Node\* temp = deque->rear; int deletedValue = temp->data; deque->rear = deque->rear->prev; if (deque->rear != NULL) { deque->rear->next = NULL;

} else {

deque->front = NULL;

}

free(temp);

return deletedValue;

}

void display(struct Deque\* deque) { if (isEmpty(deque)) { printf("Deque is empty.\n"); return;

}

struct Node\* temp = deque->front; printf("Deque elements: ");

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

}

printf("\n");

}

int main() {

struct Deque deque; initDeque(&deque); insertRear(&deque, 10);

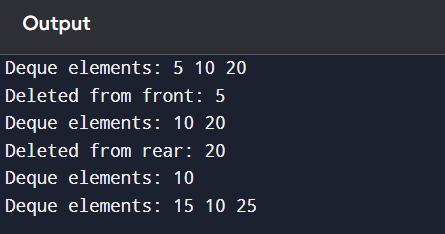
insertRear(&deque, 20);

insertFront(&deque, 5); display(&deque);

printf("Deleted from front: %d\n", deleteFront(&deque)); display(&deque); printf("Deleted from rear: %d\n", deleteRear(&deque)); display(&deque); insertFront(&deque, 15);

insertRear(&deque, 25); display(&deque);

return 0;}

**OUTPUT:**

**37.** **Write a program of Static implementation of priority queue.**

#include <stdio.h>

#define SIZE 5

int queue[SIZE];

int priority[SIZE];

int count = 0;

void enqueue(int value, int prio) {

if (count == SIZE) {

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

return; }

int i = count - 1;

while (i >= 0 && priority[i] < prio) {

queue[i + 1] = queue[i];

priority[i + 1] = priority[i];

i--; }

queue[i + 1] = value;

priority[i + 1] = prio;

count++;

printf("%d inserted with priority %d.\n", value, prio);}

void dequeue() {

if (count == 0) {

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

return;}

printf("%d with priority %d removed.\n", queue[0], priority[0]);

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

queue[i] = queue[i + 1];

priority[i] = priority[i + 1]; }

count--;}

void display() {

if (count == 0) {

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

return; }

printf("Priority Queue:\n");

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

printf("Value: %d, Priority: %d\n", queue[i], priority[i]);}}

int main() {

enqueue(10, 2);

enqueue(5, 1);

enqueue(20, 3);

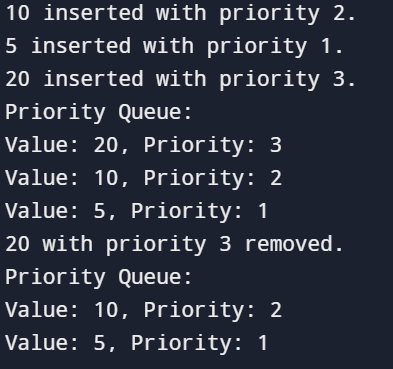
display();

dequeue();

display();

return 0;}

**OUTPUT:**

****

**38. Write a program of Dynamic implementation of priority queue.**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

int priority;

struct Node\* next;} Node;

Node\* front = NULL;

Node\* createNode(int value, int priority) {

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

temp->data = value;

temp->priority = priority;

temp->next = NULL;

return temp;}

void enqueue(int value, int priority) {

Node\* newNode = createNode(value, priority);

if (front == NULL || priority > front->priority) {

newNode->next = front;

front = newNode;

} else {

Node\* temp = front;

while (temp->next != NULL && temp->next->priority >= priority)

temp = temp->next;

newNode->next = temp->next;

temp->next = newNode;}

printf("%d inserted with priority %d.\n", value, priority);}

void dequeue() {

if (front == NULL) {

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

return;}

Node\* temp = front;

printf("%d with priority %d removed.\n", temp->data, temp->priority);

front = front->next;

free(temp);

}

void display() {

if (front == NULL) {

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

return; }

Node\* temp = front;

printf("Priority Queue:\n");

while (temp != NULL) {

printf("Value: %d, Priority: %d\n", temp->data, temp->priority);

temp = temp->next; }}

int main() {

enqueue(10, 2);

enqueue(5, 1);

enqueue(20, 3);

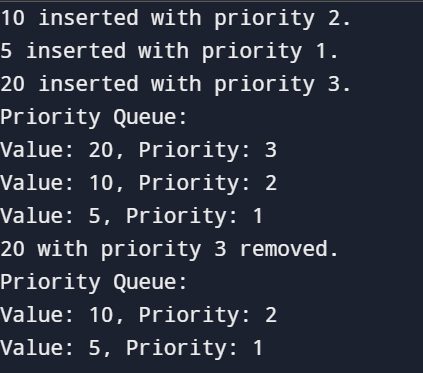
display();

dequeue();

display();

return 0;}

**OUTPUT:**

****

**39. Write a program for creating a binary tree and perform the following operations: • Insert • search • Preorder traversing • Postorder traversing • inorder traversing • delete • min element • max element**

#include <stdio.h> #include <stdlib.h> struct Node {

int data;

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

};

struct Node\* createNode(int data) {

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

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

}

struct Node\* insert(struct Node\* root, int data) { if (root == NULL) return createNode(data);

if (data < root->data) root->left = insert(root->left, data); else root->right = insert(root->right, data);

return root;

}

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

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

return data < root->data ? search(root->left, data) : search(root->right, data);

}

void preorder(struct Node\* root) { if (root != NULL) {

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

void inorder(struct Node\* root) { if (root != NULL) {

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

}}

void postorder(struct Node\* root) { if (root != NULL) {

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

}}

int findMin(struct Node\* root) { if (root == NULL) return -1;

while (root->left != NULL) root = root->left; return root->data;

}

int findMax(struct Node\* root) { if (root == NULL) return -1;

while (root->right != NULL) root = root->right; return root->data;

}

struct Node\* deleteNode(struct Node\* root, int data) { if (root == NULL) return root;

if (data < root->data) root->left = deleteNode(root->left, data);

else if (data > root->data) root->right = deleteNode(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 = root->right;

while (temp->left != NULL) temp = temp->left; root->data = temp->data;

root->right = deleteNode(root->right, temp->data);

}

return root;

}

int main() {

struct Node\* root = NULL; root = insert(root, 50); insert(root, 30);

insert(root, 20);

insert(root, 40);

insert(root, 70);

insert(root, 60);

insert(root, 80);

printf("Preorder traversal: "); preorder(root); printf("\nInorder traversal: "); inorder(root); printf("\nPostorder traversal: "); postorder(root);

int searchValue = 40;

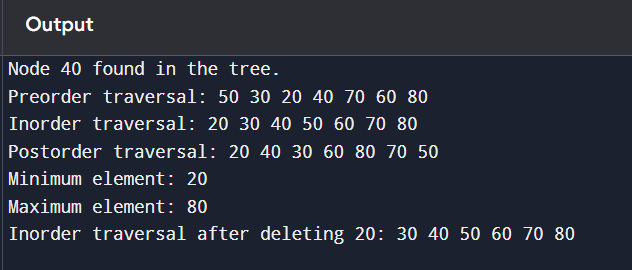
struct Node\* searchResult = search(root, searchValue);

printf("\n\nNode %d %s", searchValue, searchResult ? "found" : "not found");

printf("\nMinimum value: %d", findMin(root)); printf("\nMaximum value: %d", findMax(root));

root = deleteNode(root, 20); printf("\n\nInorder after deleting 20: "); inorder(root);

return 0;

}

**OUTPUT:**