**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");

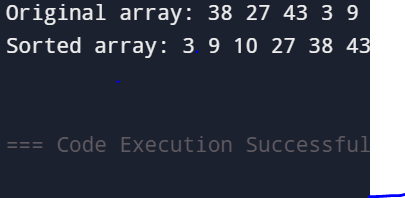
mergeSort(arr, 0, size - 1);

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; }

}

}

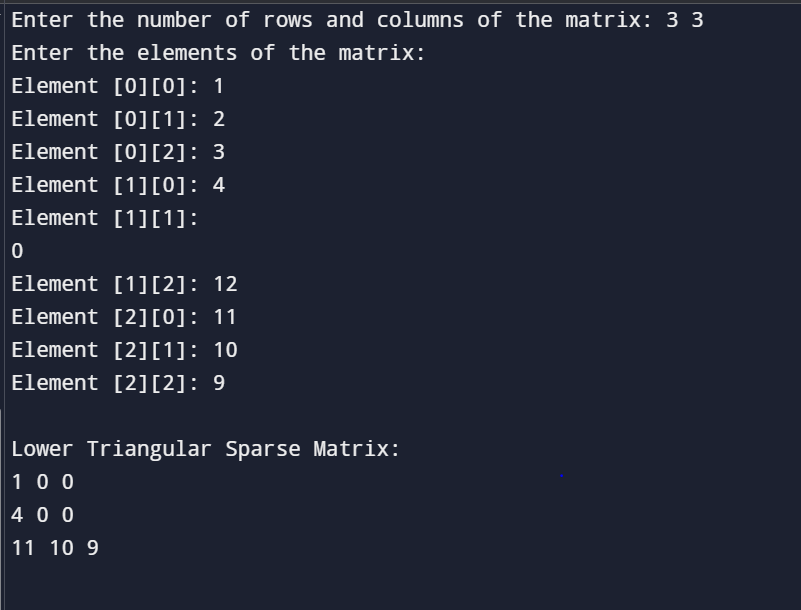
printf("\nLower Triangular Sparse Matrix:\n");

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;

}

}

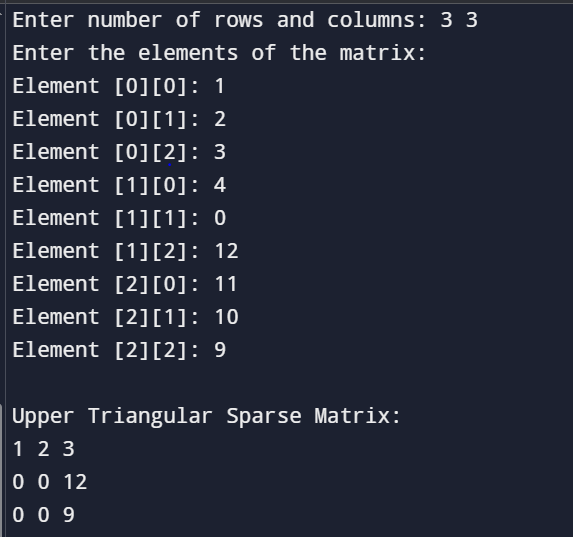
}

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 {

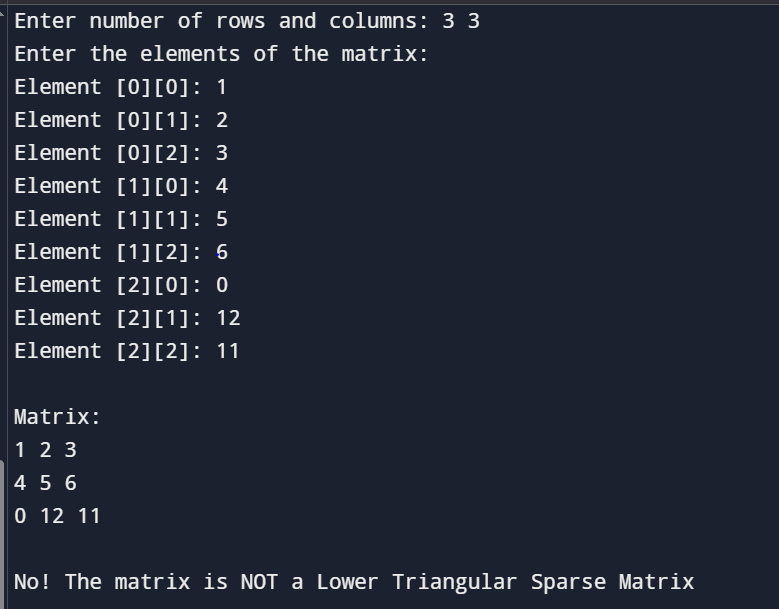
printf("\nNo! The matrix is NOT a Lower Triangular Sparse Matrix ❌\n");

}

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 {

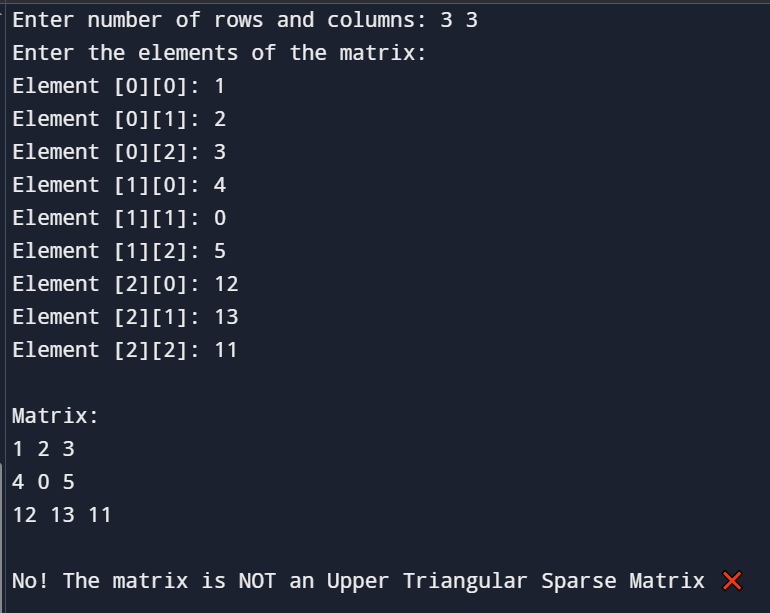
printf("\nNo! The matrix is NOT an Upper Triangular Sparse Matrix ❌\n");

}

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);

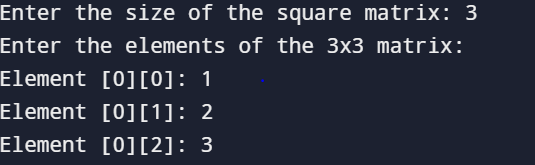
printf("\nTridiagonal Matrix:\n");

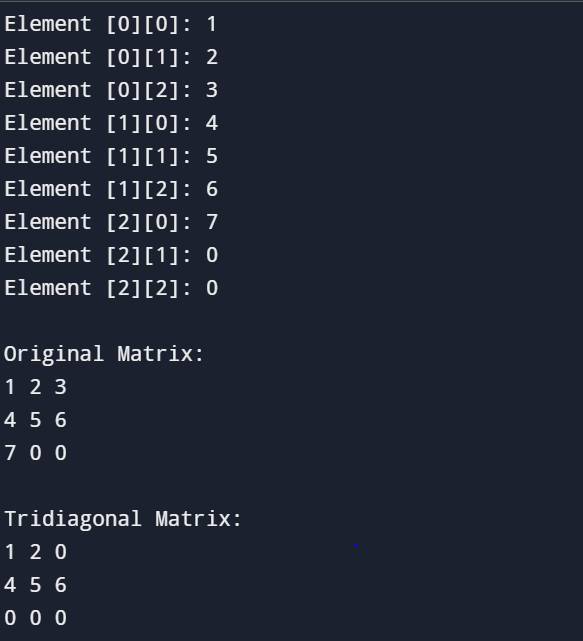
printMatrix(tridiagonal, size);

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++) {

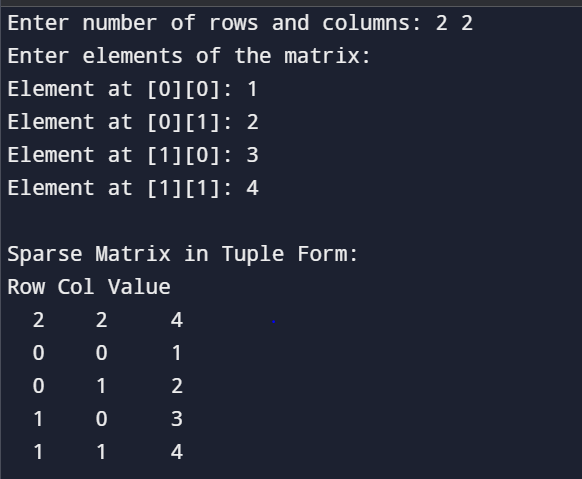
printf("%3d %4d %5d\n", tuple[i][0], tuple[i][1], tuple[i][2]);

}

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++) {

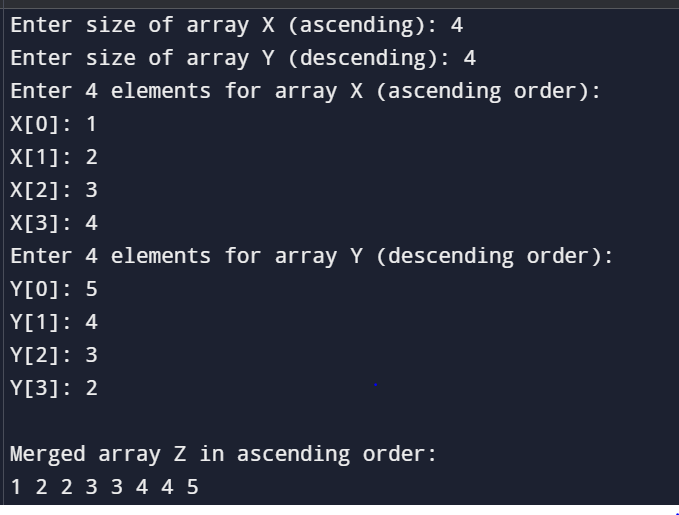
printf("%d ", Z[i]);

}

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");

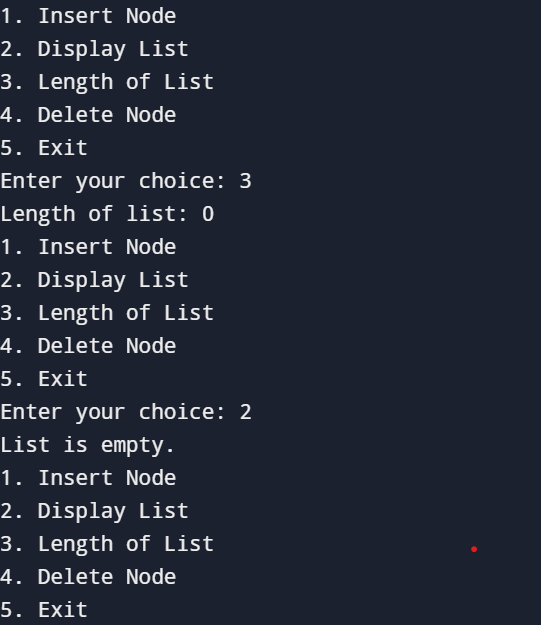
}

}

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;

}

// Delete a node with a given value

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");

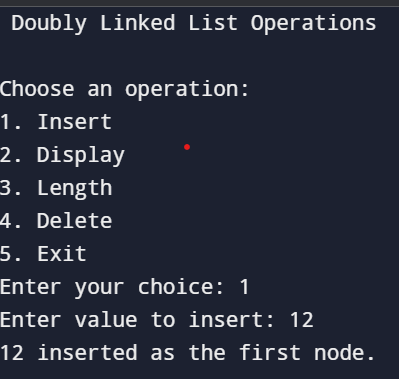
}

}

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");

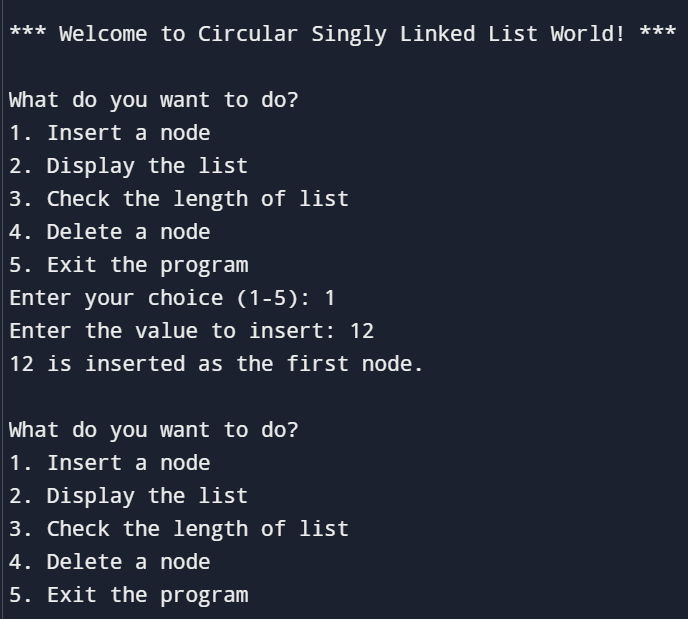
}

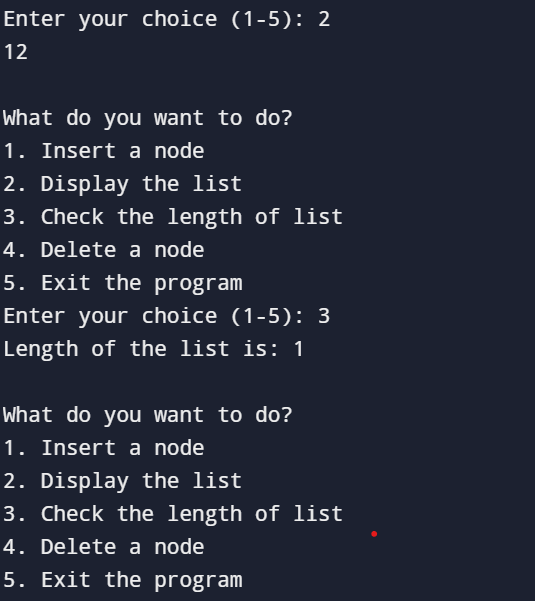
}

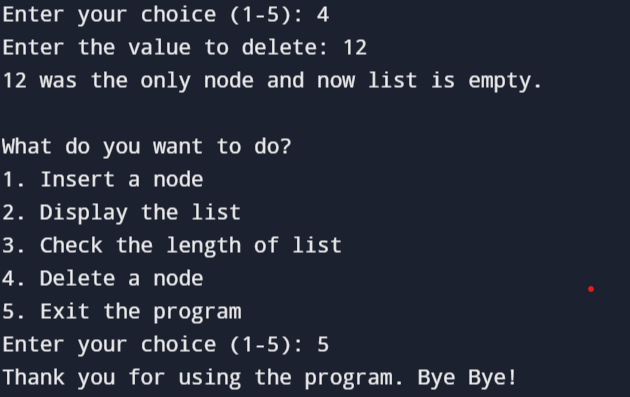
return 0;

}

**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");

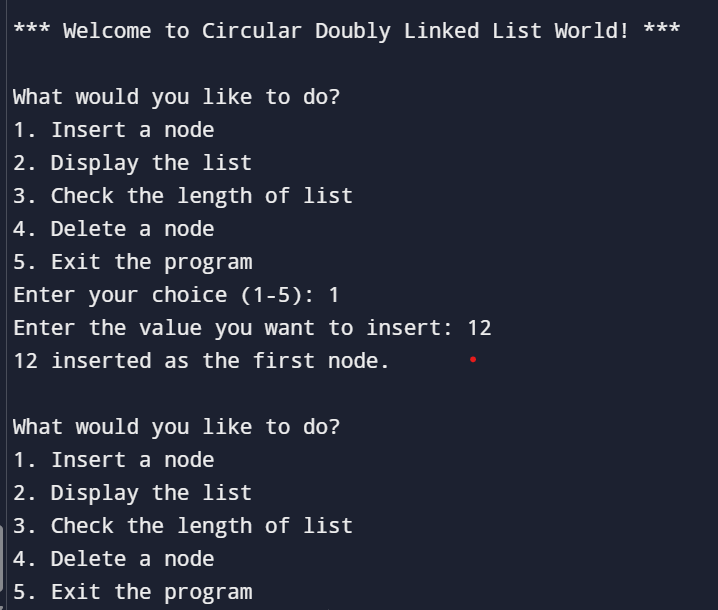
}

}

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");

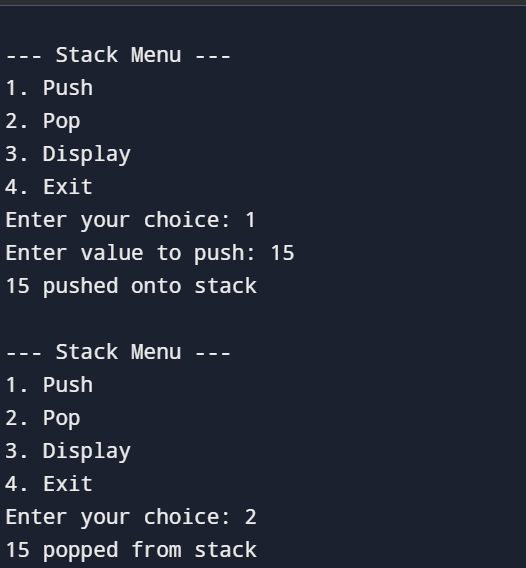
}

}

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");

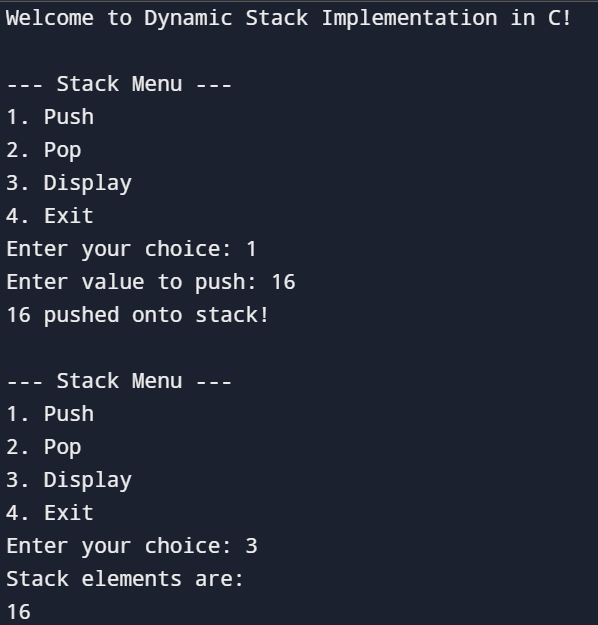
}

}

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);

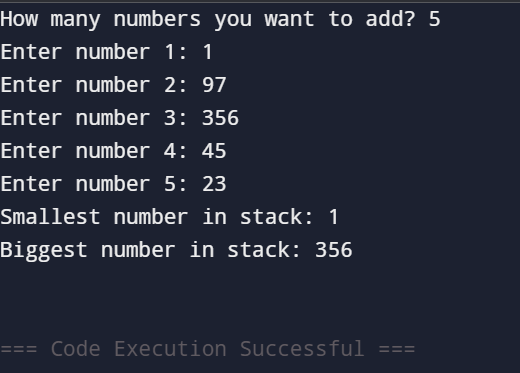
}

findSmallBig(&myStack);

return 0;

}

**OUTPUT:**

****