**1.sum of row and column in array**

**#include <stdio.h>**

**#define ROWS 3**

**#define COLS 3**

**void sumRowsAndColumns(int arr[ROWS][COLS]) {**

**int rowSum[ROWS] = {0};**

**int colSum[COLS] = {0};**

**for (int i = 0; i < ROWS; i++) {**

**for (int j = 0; j < COLS; j++) {**

**rowSum[i] += arr[i][j];**

**colSum[j] += arr[i][j];**

**}**

**}**

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

**for (int i = 0; i < ROWS; i++) {**

**printf("Row %d: %d\n", i + 1, rowSum[i]);**

**}**

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

**for (int j = 0; j < COLS; j++) {**

**printf("Column %d: %d\n", j + 1, colSum[j]);**

**}**

**}**

**int main() {**

**int arr[ROWS][COLS] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};**

**sumRowsAndColumns(arr);**

**return 0;**

**}**

**output:**



**2.Elements repeated twice -array**

**#include <stdio.h>**

**void findElementsRepeatedTwice(int arr[], int size) {**

**int hashmap[1000] = {0};**

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

**hashmap[arr[i]]++;**

**}**

**printf("Elements repeated exactly twice: ");**

**for (int i = 0; i < 1000; i++) {**

**if (hashmap[i] == 2) {**

**printf("%d ", i);**

**}**

**}**

**printf("\n");**

**}**

**int main() {**

**int arr[] = {1, 2, 3, 4, 2, 3, 5, 6, 1, 7};**

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

**findElementsRepeatedTwice(arr, size);**

**return 0;**

**}**

**output:**



**3.write c program to perform matrix multipilcation**

**#include <stdio.h>**

**#define ROW1 3**

**#define COL1 3**

**#define ROW2 3**

**#define COL2 2**

**void multiplyMatrices(int mat1[][COL1], int mat2[][COL2], int result[][COL2]) {**

**for (int i = 0; i < ROW1; i++) {**

**for (int j = 0; j < COL2; j++) {**

**result[i][j] = 0;**

**}**

**}**

**for (int i = 0; i < ROW1; i++) {**

**for (int j = 0; j < COL2; j++) {**

**for (int k = 0; k < COL1; k++) {**

**result[i][j] += mat1[i][k] \* mat2[k][j];**

**}**

**}**

**}**

**}**

**void displayMatrix(int matrix[][COL2], int rows, int cols) {**

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

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

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

**}**

**printf("\n");**

**}**

**}**

**int main() {**

**int matrix1[ROW1][COL1] = {{1, 2, 3}, {4, 5, 6},{7, 8, 9}};**

**int matrix2[ROW2][COL2] = {{1, 2},{3, 4},{5, 6}};**

**int result[ROW1][COL2];**

**multiplyMatrices(matrix1, matrix2, result);**

**printf("Result of matrix multiplication:\n");**

**displayMatrix(result, ROW1, COL2);**

**return 0;**

**}**

**output:**



**4.write c program to find factorial of a given number without using recursion**

**#include <stdio.h>**

**unsigned long long factorial(int n) {**

**unsigned long long fact = 1;**

**if (n == 0)**

**return 1;**

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

**fact \*= i;**

**}**

**return fact;**

**}**

**int main() {**

**int num;**

**printf("Enter a number: ");**

**scanf("%d", &num);**

**if (num < 0) {**

**printf("Factorial of negative number is not defined.\n");**

**} else {**

**unsigned long long fact = factorial(num);**

**printf("Factorial of %d = %llu\n", num, fact);**

**}**

**return 0;**

**}**

**output:**



**5.write a c program to find fibonacci eries without using recursion**

**#include <stdio.h>**

**void fibonacci(int n) {**

**int a = 0, b = 1, nextTerm;**

**printf("Fibonacci Series up to %d terms:\n", n);**

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

**printf("%d, ", a);**

**nextTerm = a + b;**

**a = b;**

**b = nextTerm;**

**}**

**}**

**int main() {**

**int terms;**

**printf("Enter the number of terms: ");**

**scanf("%d", &terms);**

**fibonacci(terms);**

**return 0;**

**}**

**output:**



**6.write a c program to find factorial of a given number using recursion**

**#include <stdio.h>**

**unsigned long long factorial(int n) {**

**if (n == 0)**

**return 1;**

**else**

**return n \* factorial(n - 1);**

**}**

**int main() {**

**int num;**

**printf("Enter a number: ");**

**scanf("%d", &num);**

**if (num < 0) {**

**printf("Factorial of negative number is not defined.\n");**

**} else {**

**unsigned long long fact = factorial(num);**

**printf("Factorial of %d = %llu\n", num, fact);**

**}**

**return 0;**

**}**

**output:**



**7.Fibonacci series using recusion:**

**#include <stdio.h>**

**int fibonacci(int n) {**

**if (n <= 1)**

**return n;**

**else**

**return fibonacci(n - 1) + fibonacci(n - 2);**

**}**

**int main() {**

**int terms;**

**printf("Enter the number of terms: ");**

**scanf("%d", &terms);**

**printf("Fibonacci Series up to %d terms:\n", terms);**

**for (int i = 0; i < terms; ++i) {**

**printf("%d ", fibonacci(i));**

**}**

**printf("\n");**

**return 0;**

**}**

**Output:**



**8.Implement Array operations such as Insert,Delete and Display:**

**#include <stdio.h>**

**#define MAX\_SIZE 100**

**void display(int arr[], int size) {**

**printf("Array elements: ");**

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

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

**}**

**printf("\n");**

**}**

**void insert(int arr[], int \*size, int element) {**

**if (\*size < MAX\_SIZE) {**

**arr[\*size] = element;**

**(\*size)++;**

**} else {**

**printf("Array Overflow! Cannot insert more elements.\n");**

**}**

**}**

**void delete(int arr[], int \*size, int position) {**

**if (\*size <= 0) {**

**printf("Array Underflow! Cannot delete elements from empty array.\n");**

**return;**

**}**

**for (int i = position; i < \*size - 1; ++i) {**

**arr[i] = arr[i + 1];**

**}**

**(\*size)--;**

**}**

**int main() {**

**int arr[MAX\_SIZE] = {1, 2, 3, 4, 5};**

**int size = 5;**

**display(arr, size);**

**insert(arr, &size, 10);**

**display(arr, size);**

**delete(arr, &size, 2);**

**display(arr, size);**

**return 0;**

**}**

**Output:**



**9.implement singly linked list:**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct Node {**

**int data;**

**struct Node \*next;**

**};**

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

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

**if (newNode == NULL) {**

**printf("Memory allocation failed!\n");**

**return head;**

**}**

**newNode->data = value;**

**newNode->next = head;**

**return newNode;**

**}**

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

**struct Node \*current = head;**

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

**while (current != NULL) {**

**printf("%d -> ", current->data);**

**current = current->next;**

**}**

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

**}**

**int main() {**

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

**head = insertAtBeginning(head, 10);**

**head = insertAtBeginning(head, 20);**

**head = insertAtBeginning(head, 30);**

**display(head);**

**return 0;**

**}**

**Output:**



**10.Implement doubly linked list:**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct Node {**

**int data;**

**struct Node \*prev;**

**struct Node \*next;**

**};**

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

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

**if (newNode == NULL) {**

**printf("Memory allocation failed!\n");**

**return head;**

**}**

**newNode->data = value;**

**newNode->prev = NULL;**

**newNode->next = head;**

**if (head != NULL) {**

**head->prev = newNode;**

**}**

**return newNode;**

**}**

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

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

**while (head != NULL) {**

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

**head = head->next;**

**}**

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

**}**

**int main() {**

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

**head = insertAtBeginning(head, 10);**

**head = insertAtBeginning(head, 20);**

**head = insertAtBeginning(head, 30);**

**display(head);**

**return 0;**

**}**

**Output:**



**11.Implement circular linked list:**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct Node {**

**int data;**

**struct Node \*next;**

**};**

**struct Node\* insertAtBeginning(struct Node \*last, int value) {**

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

**if (newNode == NULL) {**

**printf("Memory allocation failed!\n");**

**return last;**

**}**

**newNode->data = value;**

**if (last == NULL) {**

**newNode->next = newNode;**

**return newNode;**

**} else {**

**newNode->next = last->next;**

**last->next = newNode;**

**return last;**

**}**

**}**

**void display(struct Node \*last) {**

**if (last == NULL) {**

**printf("Circular Linked List is empty.\n");**

**return;**

**}**

**struct Node \*first = last->next;**

**printf("Circular Linked List: ");**

**struct Node \*current = first;**

**do {**

**printf("%d -> ", current->data);**

**current = current->next;**

**} while (current != first);**

**printf("...\n");**

**}**

**int main() {**

**struct Node \*last = NULL;**

**last = insertAtBeginning(last, 10);**

**last = insertAtBeginning(last, 20);**

**last = insertAtBeginning(last, 30);**

**display(last);**

**return 0;**

**}**

**Output:**

