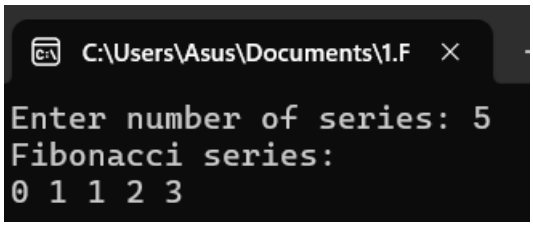
DAA Programs

Fibonacci series using Recursion.

```
#include<stdio.h>
int fibonacciSeries(int n){
	if(n<=1)
	return n;
	return fibonacciSeries(n-1)+fibonacciSeries(n-2);
}
int main(){
	int n,i;
	printf("Enter number of series: ");
	scanf("%d",&n);
	printf("Fibonacci series:\n");
	for(i=0;i<n;i++){
		printf("%d ",fibonacciSeries(i));
	}
	return 0;
}
```

Output:

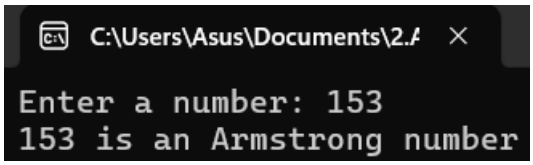


Armstrong Number.

```
#include<stdio.h>
#include<math.h>
int main(){
	int n,sum=0,rem,temp;
	printf("Enter a number: ");
```

```
    scanf("%d",&n);
    temp=n;
    while(n>0){
        rem=n%10;
        sum=sum+pow(rem,3);
        n/=10;
    }
    if(temp==sum)
    printf("%d is an Armstrong number",temp);
    else
    printf("%d is not an Armstrong number",temp);
    return 0;
}
```

Output:

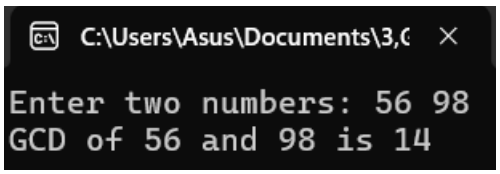


GCD of two numbers.

```
#include<stdio.h>
int main(){
    int a,b,rem;
    printf("Enter two numbers: ");
    scanf("%d %d",&a,&b);
    int n1=a;
    int n2=b;
    while(b!=0){
        rem=a%b;
        a=b;
        b=rem;
```

```
    }
    printf("GCD of %d and %d is %d",n1,n2,a);
}
```

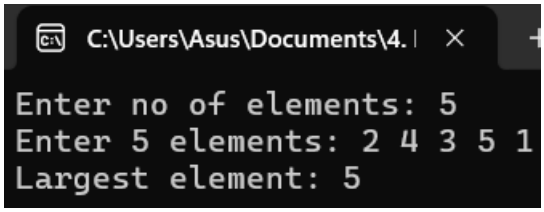
Output:



4. Largest element in an array.

```
#include<stdio.h>
int main(){
    int n,i;
    printf("Enter no of elements: ");
    scanf("%d",&n);
    int arr[n];
    printf("Enter %d elements: ",n);
    for(i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
    int max=arr[0];
    for(i=0;i<n;i++){
        if(arr[i]>max)
        max=arr[i];
    }
    printf("Largest element: %d",max);
    return 0;
}
```

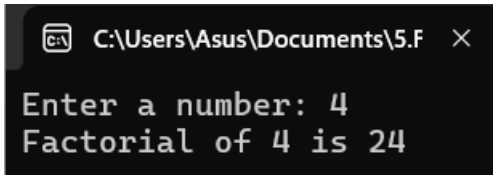
Output:



5. Factorial of a number.

```
#include<stdio.h>
int main(){
	int n,i,fact=1;
	printf("Enter a number: ");
	scanf("%d",&n);
	for(i=1;i<=n;i++){
		fact=fact*i;
	}
	printf("Factorial of %d is %d",n,fact);
	return 0;
}
```

Output:

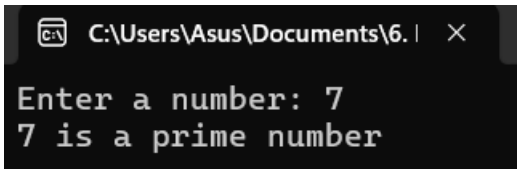


6. Prime number.

```
#include<stdio.h>
int main(){
	int n,i,count=0;
	printf("Enter a number: ");
	scanf("%d",&n);
	for(i=1;i<=n;i++){
		if(n%i==0)
```

```
            count++;
        }
        if(count==2)
        printf("%d is a prime number",n);
        else
        printf("%d is not a prime number",n);
        return 0;
}
```

Output:

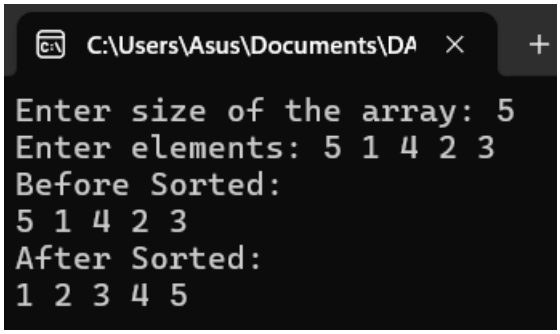


7. Selection Sort.

```
#include<stdio.h>
void selectionSort(int arr[],int n){
        int i,j,minIndex,temp;
        for(i=0;i<n;i++){
                minIndex=i;
                for(j=i+1;j<n;j++){
                        if(arr[j]<arr[minIndex]){
                                minIndex=j;
                        }
                }
                if(minIndex!=i){
                        temp=arr[i];
                        arr[i]=arr[minIndex];
                        arr[minIndex]=temp;
                }
        }
```

```
}
void printArray(int arr[],int n){
	int i;
	for(i=0;i<n;i++){
		printf("%d ",arr[i]);
	}
	printf("\n");
}
int main(){
	int i,n;
	printf("Enter size of the array: ");
	scanf("%d",&n);
	int arr[n];
	printf("Enter elements: ");
	for(i=0;i<n;i++){
		scanf("%d",&arr[i]);
	}
	printf("Before Sorted:\n");
	printArray(arr,n);
	selectionSort(arr,n);
	printf("After Sorted:\n");
	printArray(arr,n);
	return 0;
}
```

Output:

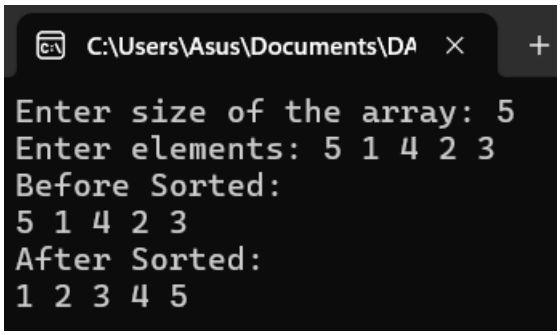


8. Bubble Sort.

```
#include<stdio.h>
void bubbleSort(int arr[],int n){
	int i,j,temp;
	for(i=0;i<n;i++){
		for(j=0;j<n-1;j++){
			if(arr[j]>arr[j+1]){
				temp=arr[j];
				arr[j]=arr[j+1];
				arr[j+1]=temp;
			}
		}
	}
}
void printArray(int arr[],int n){
	int i;
	for(i=0;i<n;i++){
		printf("%d ",arr[i]);
	}
	printf("\n");
}
int main(){
	int i,n;
	printf("Enter size of the array: ");
	scanf("%d",&n);
	int arr[n];
	printf("Enter elements: ");
	for(i=0;i<n;i++){
		scanf("%d",&arr[i]);
	}
```

```
    printf("Before Sorted:\n");
    printArray(arr,n);
    bubbleSort(arr,n);
    printf("After Sorted:\n");
    printArray(arr,n);
    return 0;
}
```

Output:

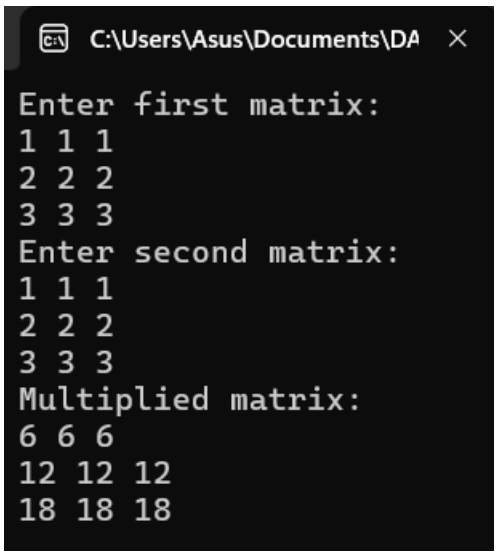


9. Matrix Multiplication.

```
#include<stdio.h>
int main(){
    int a[3][3],b[3][3],c[3][3];
    int i,j,k;
    printf("Enter first matrix: \n");
    for(i=0;i<3;i++){
        for(j=0;j<3;j++){
            scanf("%d",&a[i][j]);
        }
    }
    printf("Enter second matrix: \n");
    for(i=0;i<3;i++){
        for(j=0;j<3;j++){
            scanf("%d",&b[i][j]);
```

```
            }
        }
        for(i=0;i<3;i++){
            for(j=0;j<3;j++){
                c[i][j]=0;
                for(k=0;k<3;k++){
                    c[i][j]+=a[i][k]*b[k][j];
                }
            }
        }
        printf("Multiplied matrix:\n");
        for(i=0;i<3;i++){
            for(j=0;j<3;j++){
                printf("%d ",c[i][j]);
            }
            printf("\n");
        }
        return 0;
}
```

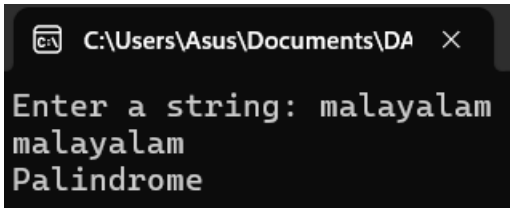
Output:



10. String Palindrome.

```
#include<stdio.h>
#include<string.h>
int main(){
	char str[100],temp,ori[100];
	printf("Enter a string: ");
	scanf("%s",str);
	strcpy(ori,str);
	int l=strlen(str);
	int s,e;
	s=0;
	e=l-1;
	while(s<e){
		temp=str[s];
		str[s]=str[e];
		str[e]=temp;
		s++;
		e--;
	}
	printf("%s\n",str);
	if(strcmp(ori,str)==0){
		printf("Palindrome");
	}
	else{
		printf("Not Palindrome");
	}
	return 0;
}
```

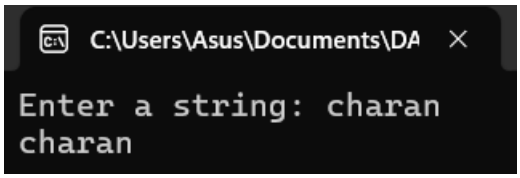
Output:



11. Copy String.

```
#include <stdio.h>
#include<string.h>
int main() {
    char str[100], ori[100];
    int i;
    printf("Enter a string: ");
    scanf("%s",str);
        for(i=0;i<strlen(str);i++){
                ori[i]=str[i];
        }
        printf("%s",ori);
    return 0;
}
```

Output:



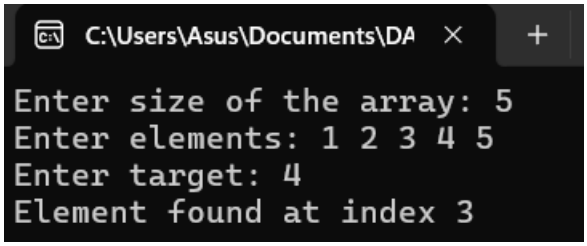
12. Binary Search.

```
#include<stdio.h>
int binarySearch(int arr[],int n,int target){
        int low=0,high=n-1;
        while(low<=high){
                int mid=(low+high)/2;
```

```
            if(arr[mid]==target){
                return mid;
            }
            if(arr[mid]<target){
                low=mid+1;
            }
            else{
                high=mid-1;
            }
        }
        return -1;
}
int main(){
        int i,n,target;
        printf("Enter size of the array: ");
        scanf("%d",&n);
        int arr[n];
        printf("Enter elements: ");
        for(i=0;i<n;i++){
            scanf("%d",&arr[i]);
        }
        printf("Enter target: ");
        scanf("%d",&target);
        int result=binarySearch(arr,n,target);
        if(result!=-1){
            printf("Element found at index %d",result);
        }
        else{
            printf("Element not found");
        }
```

```
    return 0;
}
```

Output:

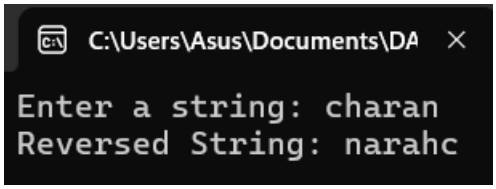


13. Reverse String.

```
#include<stdio.h>
#include<string.h>
int main(){
    char str[100],temp,ori[100];
    printf("Enter a string: ");
    scanf("%s",str);
    strcpy(ori,str);
    int l=strlen(str);
    int s,e;
    s=0;
    e=l-1;
    while(s<e){
        temp=str[s];
        str[s]=str[e];
        str[e]=temp;
        s++;
        e--;
    }
    printf("Reversed String: %s\n",str);

    return 0;
```

```
}
```

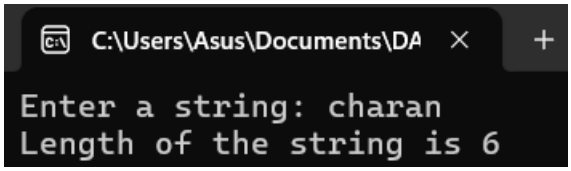
Output:



14. String length.

```
#include<stdio.h>
int main(){
	char str[100];
	int i,count=0;
	printf("Enter a string: ");
	scanf("%s",str);
	for(i=0;str[i]!='\0';i++){
		if(str[i]!='\n'){
			count++;
		}
	}
	printf("Length of the string is %d",count);
	return 0;
}
```

Output:



15. Strassen's Matrix.

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

```
#define MAX 4  // Matrix size 4x4

// Function to add matrices
void add(int A[MAX][MAX], int B[MAX][MAX], int C[MAX][MAX], int size) {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            C[i][j] = A[i][j] + B[i][j];
        }
    }
}

// Function to subtract matrices
void subtract(int A[MAX][MAX], int B[MAX][MAX], int C[MAX][MAX], int size) {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            C[i][j] = A[i][j] - B[i][j];
        }
    }
}

// Strassen's algorithm for matrix multiplication
void strassen(int A[MAX][MAX], int B[MAX][MAX], int C[MAX][MAX], int size) {
    if (size == 1) {
        C[0][0] = A[0][0] * B[0][0];
        return;
    }

    int newSize = size / 2;
    int A11[MAX][MAX], A12[MAX][MAX], A21[MAX][MAX], A22[MAX][MAX];
    int B11[MAX][MAX], B12[MAX][MAX], B21[MAX][MAX], B22[MAX][MAX];
```

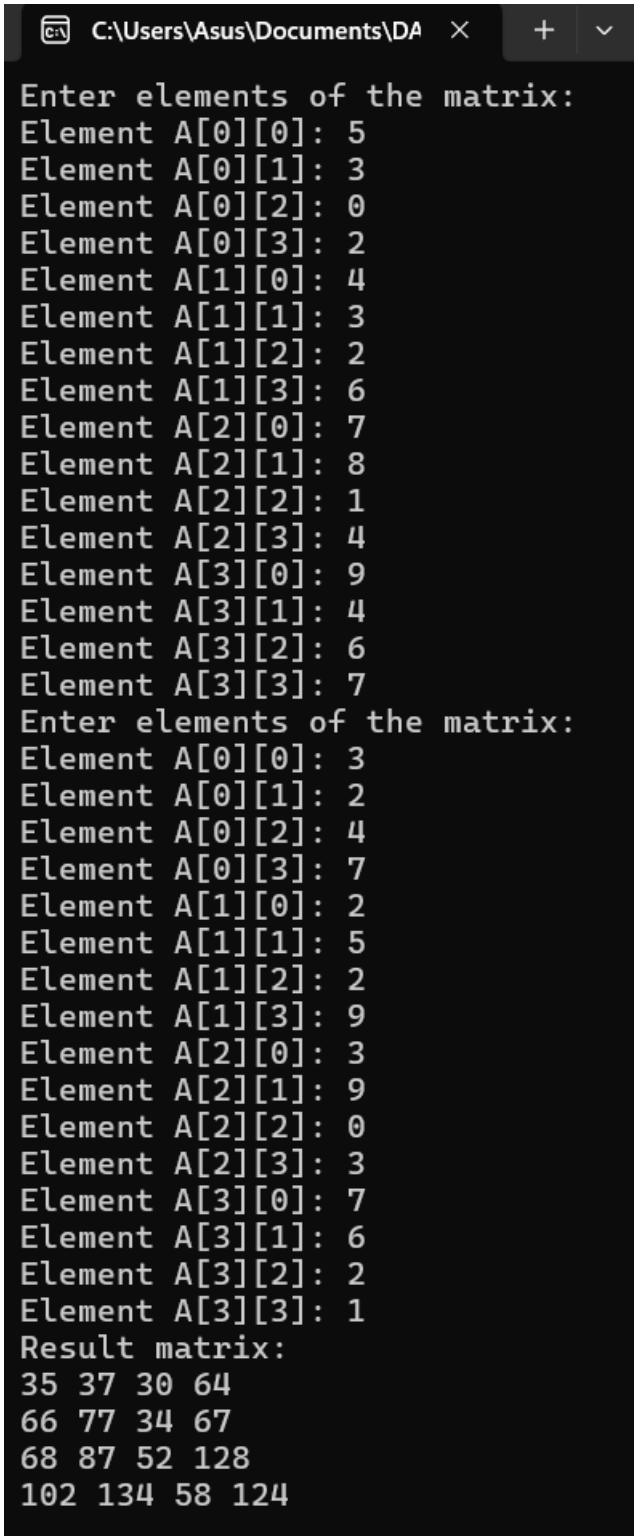
```
    int C11[MAX][MAX], C12[MAX][MAX], C21[MAX][MAX], C22[MAX][MAX];
    int M1[MAX][MAX], M2[MAX][MAX], M3[MAX][MAX], M4[MAX][MAX],
M5[MAX][MAX], M6[MAX][MAX], M7[MAX][MAX];
    int temp1[MAX][MAX], temp2[MAX][MAX];

    // Divide the matrices into submatrices
    for (int i = 0; i < newSize; i++) {
        for (int j = 0; j < newSize; j++) {
            A11[i][j] = A[i][j];
            A12[i][j] = A[i][j + newSize];
            A21[i][j] = A[i + newSize][j];
            A22[i][j] = A[i + newSize][j + newSize];

            B11[i][j] = B[i][j];
            B12[i][j] = B[i][j + newSize];
            B21[i][j] = B[i + newSize][j];
            B22[i][j] = B[i + newSize][j + newSize];
        }
    }

    // Calculate M1 to M7
    add(A11, A22, temp1, newSize);
    add(B11, B22, temp2, newSize);
    strassen(temp1, temp2, M1, newSize);

    add(A21, A22, temp1, newSize);
    strassen(temp1, B11, M2, newSize);

    subtract(B12, B22, temp2, newSize);
    strassen(A11, temp2, M3, newSize);
```

```
subtract(B21, B11, temp2, newSize);
strassen(A22, temp2, M4, newSize);

add(A11, A12, temp1, newSize);
strassen(temp1, B22, M5, newSize);

subtract(A21, A11, temp1, newSize);
add(B11, B12, temp2, newSize);
strassen(temp1, temp2, M6, newSize);

subtract(A12, A22, temp1, newSize);
add(B21, B22, temp2, newSize);
strassen(temp1, temp2, M7, newSize);

// Calculate C11, C12, C21, C22
add(M1, M4, temp1, newSize);
subtract(temp1, M5, temp2, newSize);
add(temp2, M7, C11, newSize);

add(M3, M5, C12, newSize);

add(M2, M4, C21, newSize);

add(M1, M3, temp1, newSize);
subtract(temp1, M2, temp2, newSize);
add(temp2, M6, C22, newSize);

// Combine C11, C12, C21, C22 into C
for (int i = 0; i < newSize; i++) {
    for (int j = 0; j < newSize; j++) {
```

```
            C[i][j] = C11[i][j];
            C[i][j + newSize] = C12[i][j];
            C[i + newSize][j] = C21[i][j];
            C[i + newSize][j + newSize] = C22[i][j];
        }
    }
}

// Function to take matrix input
void inputMatrix(int A[MAX][MAX], int size) {
    printf("Enter elements of the matrix:\n");
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            printf("Element A[%d][%d]: ", i, j);
            scanf("%d", &A[i][j]);
        }
    }
}

// Function to display matrix
void displayMatrix(int A[MAX][MAX], int size) {
    printf("Result matrix:\n");
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            printf("%d ", A[i][j]);
        }
        printf("\n");
    }
}
```

```
int main() {
    int size = MAX;  // Matrix size is 4x4

    int A[MAX][MAX], B[MAX][MAX], C[MAX][MAX];

    // Input matrices A and B
    inputMatrix(A, size);
    inputMatrix(B, size);

    // Perform Strassen's matrix multiplication
    strassen(A, B, C, size);

    // Display the result
    displayMatrix(C, size);

    return 0;
}
```

Output:



16. Merge Sort.

```
#include <stdio.h>


// Function to merge two subarrays
```

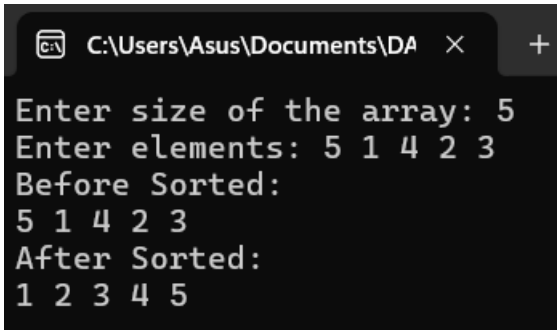
```
void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    // Create temporary arrays
    int L[n1], R[n2];

    // Copy data to temporary arrays L[] and R[]
    for (int i = 0; i < n1; i++) {
        L[i] = arr[left + i];
    }
    for (int j = 0; j < n2; j++) {
        R[j] = arr[mid + 1 + j];
    }

    // Merge the temporary arrays back into arr[left..right]
    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    // Copy the remaining elements of L[], if any
    while (i < n1) {
```

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

    // Copy the remaining elements of R[], if any
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

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

        // Recursively sort the first and second halves
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

        // Merge the sorted halves
        merge(arr, left, mid, right);
    }
}

// Function to print the array
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
```

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

// Main function to test the merge sort
int main() {
    int arrSize;

    // Get user input for array size
    printf("Enter the number of elements: ");
    scanf("%d", &arrSize);

    int arr[arrSize];

    // Get user input for array elements
    printf("Enter the elements of the array:\n");
    for (int i = 0; i < arrSize; i++) {
        scanf("%d", &arr[i]);
    }

    printf("Original array: \n");
    printArray(arr, arrSize);

    mergeSort(arr, 0, arrSize - 1);

    printf("Sorted array: \n");
    printArray(arr, arrSize);

    return 0;
```

```
}
```

Output:

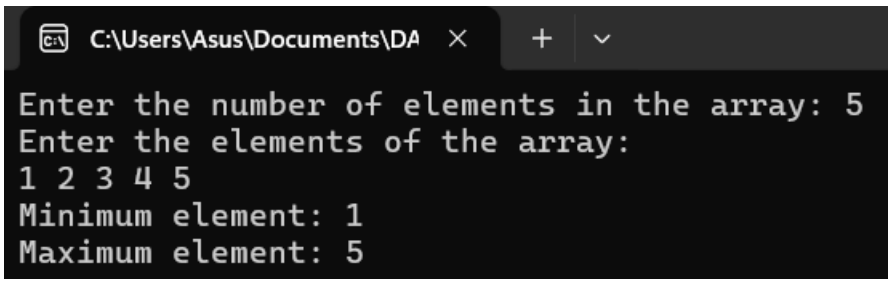


17. Min and Max elements in array.

```
#include <stdio.h>

void findMinMax(int arr[], int n) {
    int min = arr[0];
    int max = arr[0];

    for (int i = 1; i < n; i++) {
        if (arr[i] < min) {
            min = arr[i];
        }
        if (arr[i] > max) {
            max = arr[i];
        }
    }

    printf("Minimum element: %d\n", min);
    printf("Maximum element: %d\n", max);
}

int main() {
```

```
    int n;

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

    int arr[n];

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

    // Call the function to find the min and max
    findMinMax(arr, n);

    return 0;
}
```

Output:

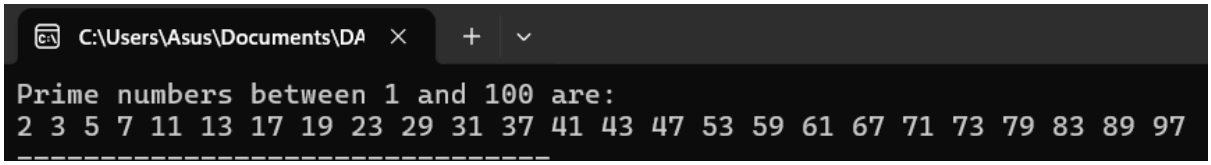


18. Prime numbers between 1 and 100.

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

```
// Function to check if a number is prime
bool isPrime(int num) {
    if (num <= 1) {
        return false; // 1 and numbers less than 1 are not prime
    }

    for (int i = 2; i * i <= num; i++) {
        if (num % i == 0) {
            return false; // Number is divisible by i, so it's not prime
        }
    }
    return true; // If no divisors were found, the number is prime
}

int main() {
    printf("Prime numbers between 1 and 100 are:\n");

    for (int i = 1; i <= 100; i++) {
        if (isPrime(i)) {
            printf("%d ", i);
        }
    }

    return 0;
}
```

Output:



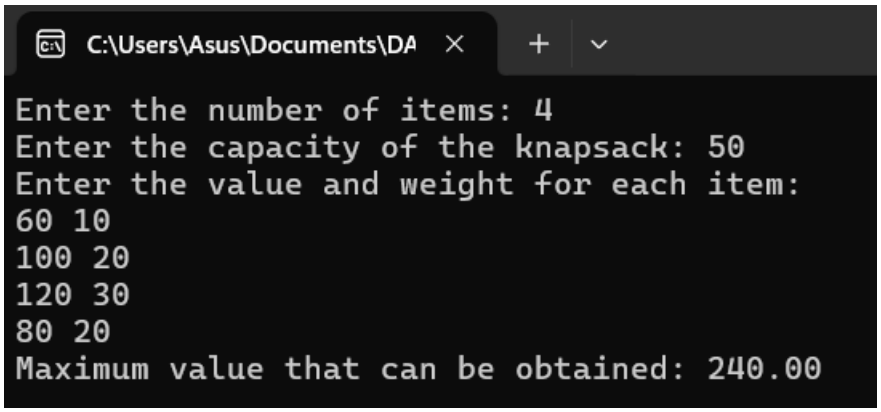
19. Knapsack using Greedy method.

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

// Structure for an item
struct Item {
    int value;
    int weight;
    float ratio; // Value-to-weight ratio
};

// Comparison function for sorting items based on value-to-weight ratio
int compare(const void *a, const void *b) {
    float ratio1 = ((struct Item*)a)->ratio;
    float ratio2 = ((struct Item*)b)->ratio;
    return (ratio2 - ratio1 > 0) - (ratio2 - ratio1 < 0);
}

// Function to solve the Fractional Knapsack problem using Greedy approach
float knapsack(struct Item items[], int n, int capacity) {
    // Sort items by value-to-weight ratio in descending order
    qsort(items, n, sizeof(struct Item), compare);

    float totalValue = 0.0;
    int remainingCapacity = capacity;

    for (int i = 0; i < n; i++) {
        if (items[i].weight <= remainingCapacity) {
            // Take the whole item
            remainingCapacity -= items[i].weight;
```

```
            totalValue += items[i].value;
        } else {
            // Take the fraction of the item that fits
            totalValue += items[i].value * ((float)remainingCapacity / items[i].weight);
            break;
        }
    }

    return totalValue;
}

int main() {
    int n, capacity;

    // Input number of items and knapsack capacity
    printf("Enter the number of items: ");
    scanf("%d", &n);

    printf("Enter the capacity of the knapsack: ");
    scanf("%d", &capacity);

    struct Item items[n];

    // Input value, weight and calculate value-to-weight ratio for each item
    printf("Enter the value and weight for each item:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d %d", &items[i].value, &items[i].weight);
        items[i].ratio = (float)items[i].value / items[i].weight;
    }
```

```
    // Calculate the maximum value that can be obtained
    float maxValue = knapsack(items, n, capacity);

    // Output the result
    printf("Maximum value that can be obtained: %.2f\n", maxValue);

    return 0;
}
```

Output:



20. MST using Greedy techniques.

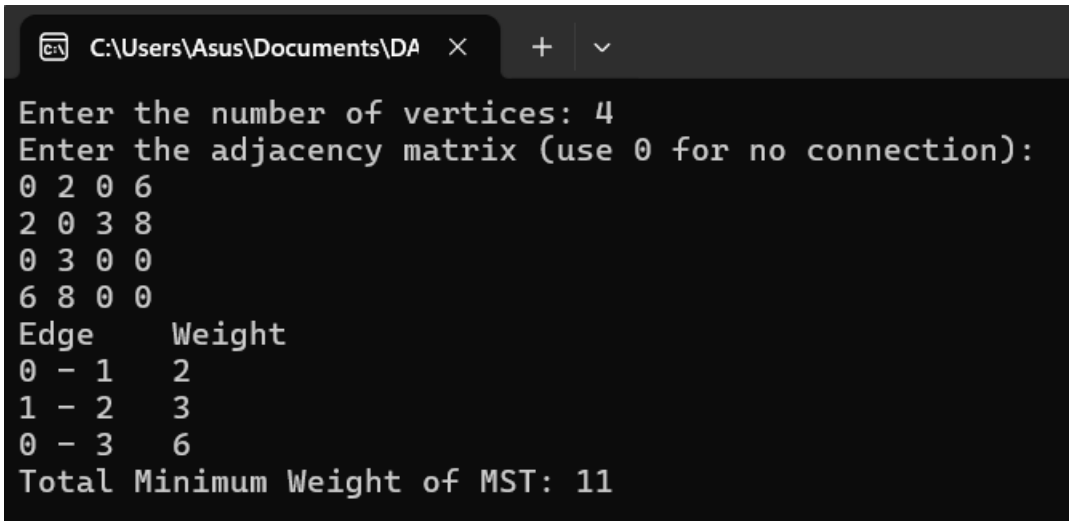
```
#include <stdio.h>
#include <limits.h>
#include <stdbool.h>

#define V 100 // Maximum number of vertices

// Function to find the vertex with the minimum key value
int minKey(int key[], bool mstSet[], int vertices) {
    int min = INT_MAX, min_index;

    for (int v = 0; v < vertices; v++) {
```

```
        if (mstSet[v] == false && key[v] < min) {
            min = key[v];
            min_index = v;
        }
    }
    return min_index;
}

// Function to print the constructed MST and calculate the total weight
void printMST(int parent[], int graph[V][V], int vertices) {
    int totalWeight = 0;
    printf("Edge \tWeight\n");
    for (int i = 1; i < vertices; i++) {
        printf("%d - %d \t%d\n", parent[i], i, graph[i][parent[i]]);
        totalWeight += graph[i][parent[i]]; // Sum the weights
    }
    printf("Total Minimum Weight of MST: %d\n", totalWeight);
}

// Function to construct and print the MST using Prim's algorithm
void primMST(int graph[V][V], int vertices) {
    int parent[V]; // Array to store the constructed MST
    int key[V];    // Key values used to pick the minimum weight edge
    bool mstSet[V]; // To represent the set of vertices included in MST

    // Initialize all keys as INFINITE
    for (int i = 0; i < vertices; i++) {
        key[i] = INT_MAX;
        mstSet[i] = false;
    }
```

```
    // Include the first vertex in MST
    key[0] = 0;
    parent[0] = -1; // First node is always the root of the MST

    for (int count = 0; count < vertices - 1; count++) {
        int u = minKey(key, mstSet, vertices);
        mstSet[u] = true;

        for (int v = 0; v < vertices; v++) {
            if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v]) {
                parent[v] = u;
                key[v] = graph[u][v];
            }
        }
    }

    printMST(parent, graph, vertices);
}

int main() {
    int vertices;
    printf("Enter the number of vertices: ");
    scanf("%d", &vertices);

    int graph[V][V];
    printf("Enter the adjacency matrix (use 0 for no connection):\n");
    for (int i = 0; i < vertices; i++) {
        for (int j = 0; j < vertices; j++) {
            scanf("%d", &graph[i][j]);
```

```
        }
    }

    primMST(graph, vertices);

    return 0;
}
```

Output:



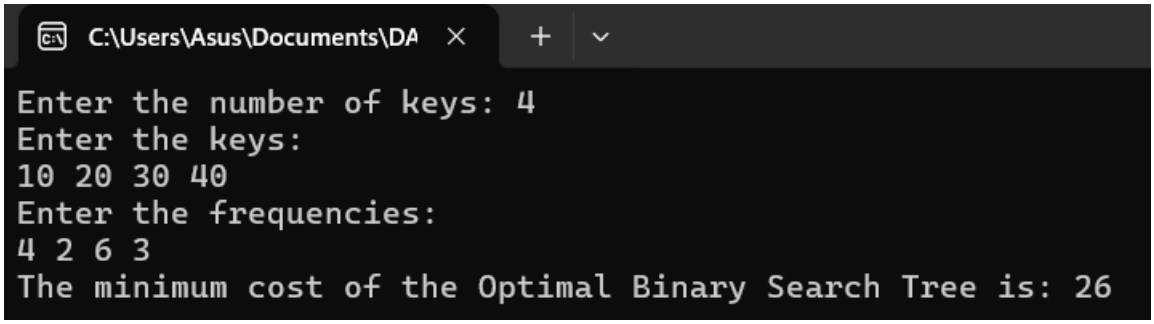
21. OBST using Dynamic Programming.

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

// Function to calculate the sum of frequencies from i to j
int sum(int freq[], int i, int j) {
    int s = 0;
    for (int k = i; k <= j; k++) {
        s += freq[k];
    }
    return s;
}
```

```
// Function to build the OBST using dynamic programming
int optimalSearchTree(int keys[], int freq[], int n) {
    int cost[n][n]; // cost[i][j] stores the minimum cost of OBST for keys[i..j]

    // Initialize the cost of single keys (single nodes)
    for (int i = 0; i < n; i++) {
        cost[i][i] = freq[i];
    }

    // Build the table for subtrees of increasing size
    for (int length = 2; length <= n; length++) {
        for (int i = 0; i <= n - length; i++) {
            int j = i + length - 1;
            cost[i][j] = INT_MAX;

            // Try making each key in keys[i..j] as the root
            for (int r = i; r <= j; r++) {
                int c = ((r > i) ? cost[i][r - 1] : 0) +
                        ((r < j) ? cost[r + 1][j] : 0) +
                        sum(freq, i, j);

                if (c < cost[i][j]) {
                    cost[i][j] = c;
                }
            }
        }
    }

    return cost[0][n - 1]; // Minimum cost of OBST for keys[0..n-1]
```

```
}

int main() {
    int n;
    printf("Enter the number of keys: ");
    scanf("%d", &n);

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

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

    int minCost = optimalSearchTree(keys, freq, n);
    printf("The minimum cost of the Optimal Binary Search Tree is: %d\n", minCost);

    return 0;
}
```

Output:

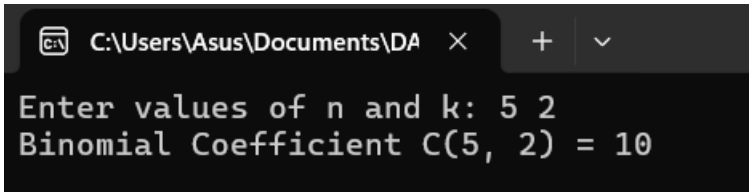


22. Binomial Coefficient.

```
#include <stdio.h>

// Function to calculate Binomial Coefficient using dynamic programming
int binomialCoeff(int n, int k) {
    int C[n+1][k+1];

    // Calculate value of Binomial Coefficient in bottom-up manner
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= (i < k ? i : k); j++) {
            // Base Case: C(i, 0) = 1 and C(i, i) = 1
            if (j == 0 || j == i) {
                C[i][j] = 1;
            } else {
                // Recursive Case: C(i, j) = C(i-1, j-1) + C(i-1, j)
                C[i][j] = C[i-1][j-1] + C[i-1][j];
            }
        }
    }

    return C[n][k]; // Return the binomial coefficient C(n, k)
}

int main() {
    int n, k;

    // Input values for n and k
    printf("Enter values of n and k: ");
    scanf("%d %d", &n, &k);
```

```
    // Output the binomial coefficient
    printf("Binomial Coefficient C(%d, %d) = %d\n", n, k, binomialCoeff(n, k));

    return 0;
}
```

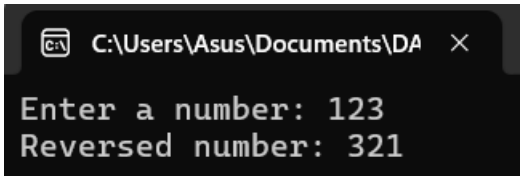
Output:



23. Reverse a number.

```
#include <stdio.h>
int main() {
    int num, reversed = 0, remainder;
    printf("Enter a number: ");
    scanf("%d", &num);
    while (num != 0) {
        remainder = num % 10;
        reversed = reversed * 10 + remainder;
        num = num / 10;
    }
    printf("Reversed number: %d\n", reversed);
    return 0;
}
```

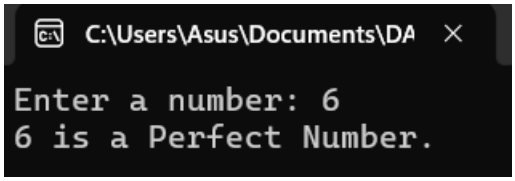
Output:



24. Perfect number.

```
#include <stdio.h>
int main() {
    int num, sum = 0;
    printf("Enter a number: ");
    scanf("%d", &num);
    for (int i = 1; i <= num / 2; i++) {
        if (num % i == 0) {
            sum += i;
        }
    }
    if (sum == num) {
        printf("%d is a Perfect Number.\n", num);
    } else {
        printf("%d is not a Perfect Number.\n", num);
    }
    return 0;
}
```

Output:



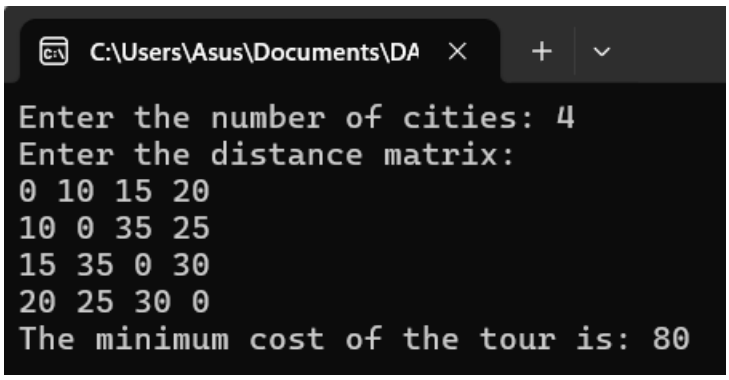
25. TSP using Dynamic Programming.

```
#include <stdio.h>

#define MAX 16
#define INF 9999999   // Define a large number to represent infinity

int dp[1 << MAX][MAX];  // DP table to store the minimum cost
```

```
int dist[MAX][MAX];     // Matrix to store distances between cities

// Function to solve the Traveling Salesman Problem using Dynamic Programming and Bitmasking
int tsp(int mask, int pos, int n) {
    if (mask == (1 << n) - 1) {  // All cities have been visited
        return dist[pos][0];  // Return to the starting city
    }

    if (dp[mask][pos] != -1)  // If the result is already calculated, return it
        return dp[mask][pos];

    int ans = INF;
    for (int city = 0; city < n; city++) {
        if ((mask & (1 << city)) == 0) {  // If the city hasn't been visited
            int newAns = dist[pos][city] + tsp(mask | (1 << city), city, n);
            ans = (ans < newAns) ? ans : newAns;  // Choose the minimum cost
        }
    }
    return dp[mask][pos] = ans;  // Store the result in DP table
}

int main() {
    int n;
    printf("Enter the number of cities: ");
    scanf("%d", &n);

    if (n > MAX) {
        printf("The maximum number of cities supported is %d.\n", MAX);
        return -1;
    }
```

```
    printf("Enter the distance matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &dist[i][j]);
        }
    }

    // Initialize DP table with -1 (meaning uncalculated)
    for (int i = 0; i < (1 << n); i++) {
        for (int j = 0; j < n; j++) {
            dp[i][j] = -1;
        }
    }

    // Calculate the result starting from city 0, with only city 0 visited (mask = 1)
    int result = tsp(1, 0, n);
    printf("The minimum cost of the tour is: %d\n", result);

    return 0;
}
```

Output:

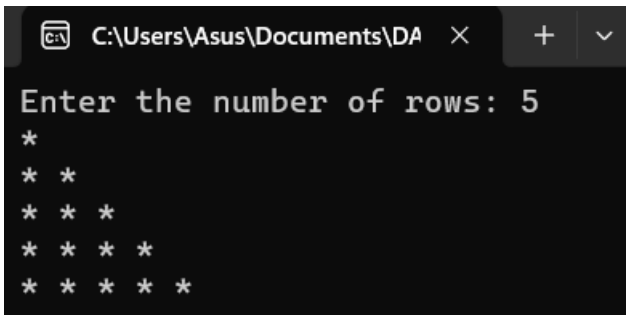


26. Print the pattern.

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

```
int main() {
    int i, j, n;

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

    for (i = 1; i <= n; i++) {
        for (j = 1; j <= i; j++) {
            printf("* ");
        }
        printf("\n");
    }

    return 0;
}
```

Output:



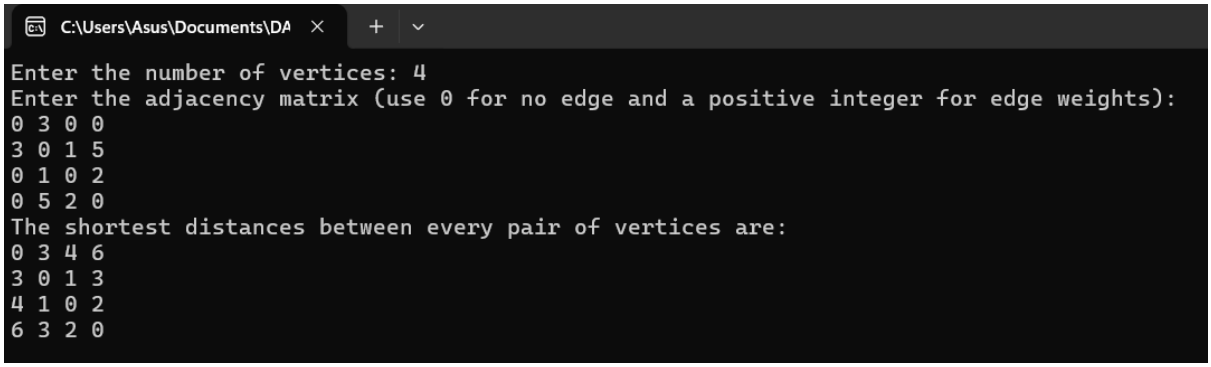
27. Floyd’s algorithm.

```
#include <stdio.h>
#define INF 9999999
#define MAX 10
void floydWarshall(int graph[MAX][MAX], int n) {
    int dist[MAX][MAX], i, j, k;
    for (i = 0; i < n; i++) {
```

```
        for (j = 0; j < n; j++) {
            if (i == j) dist[i][j] = 0;
            else if (graph[i][j] == 0) dist[i][j] = INF;
            else dist[i][j] = graph[i][j];
        }
    }
    for (k = 0; k < n; k++) {
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++) {
                if (dist[i][j] > dist[i][k] + dist[k][j]) dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }
    printf("The shortest distances between every pair of vertices are:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (dist[i][j] == INF) printf("INF ");
            else printf("%d ", dist[i][j]);
        }
        printf("\n");
    }
}
int main() {
    int n, i, j;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    int graph[MAX][MAX];
    printf("Enter the adjacency matrix (use 0 for no edge and a positive integer for
edge weights):\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
```

```
            scanf("%d", &graph[i][j]);
        }
    }
    floydWarshall(graph, n);
    return 0;
}
```

Output:

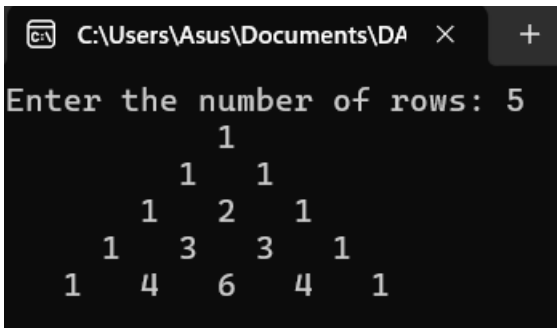


28. Pascal's Triangle.

```
#include <stdio.h>
int main() {
    int n, i, j, val, space;
    printf("Enter the number of rows: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++) {
        for (space = 0; space < n - i - 1; space++) {
            printf("  ");
        }
        val = 1;
        for (j = 0; j <= i; j++) {
            printf("%4d", val);
            val = val * (i - j) / (j + 1);
        }
        printf("\n");
```

```
    }
    return 0;
}
```

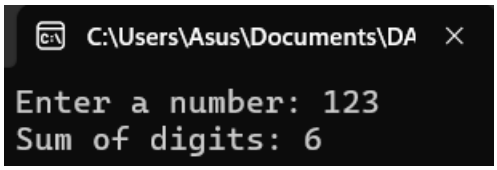
Output:



29. Sum of digits.

```
#include <stdio.h>
int main() {
    int num, sum = 0;
    printf("Enter a number: ");
    scanf("%d", &num);
    while (num != 0) {
        sum += num % 10;
        num /= 10;
    }
    printf("Sum of digits: %d\n", sum);
    return 0;
}
```

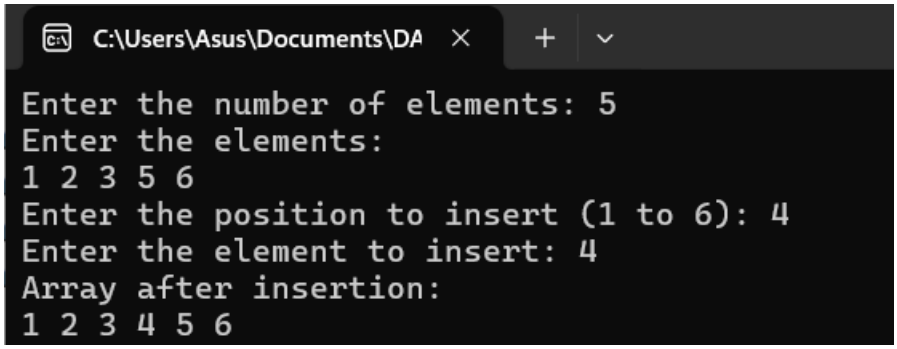
Output:



30. Inserting element in an array.

```c
#include <stdio.h>
int main() {
    int arr[100], n, pos, elem, i;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    printf("Enter the elements:\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Enter the position to insert (1 to %d): ", n + 1);
    scanf("%d", &pos);
    printf("Enter the element to insert: ");
    scanf("%d", &elem);
    for (i = n; i >= pos - 1; i--) {
        arr[i + 1] = arr[i];
    }
    arr[pos - 1] = elem;
    n++;
    printf("Array after insertion:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Output:

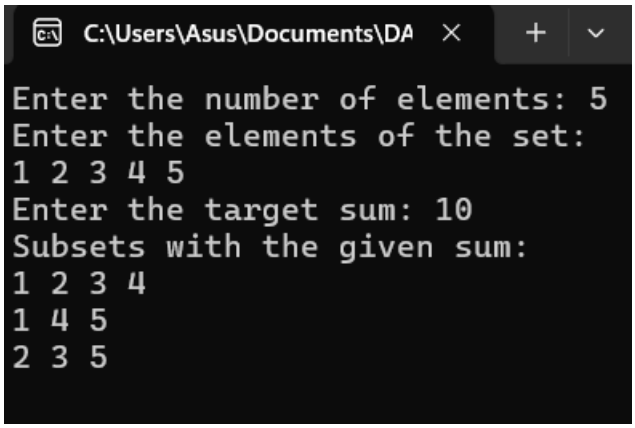


31. Sum of subsets.

```
#include <stdio.h>

int total = 0;

void subsetSum(int set[], int subset[], int n, int subsetSize, int sum, int targetSum, int
index) {
    if (sum == targetSum) {
        for (int i = 0; i < subsetSize; i++) {
            printf("%d ", subset[i]);
        }
        printf("\n");
        return;
    }

    for (int i = index; i < n; i++) {
        if (sum + set[i] <= targetSum) {
            subset[subsetSize] = set[i];
            subsetSum(set, subset, n, subsetSize + 1, sum + set[i], targetSum, i + 1);
        }
    }
}

int main() {
    int n, targetSum;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int set[n], subset[n];
    printf("Enter the elements of the set:\n");
    for (int i = 0; i < n; i++) {
```

```
        scanf("%d", &set[i]);
    }
    printf("Enter the target sum: ");
    scanf("%d", &targetSum);
    printf("Subsets with the given sum:\n");
    subsetSum(set, subset, n, 0, 0, targetSum, 0);
    return 0;
}
```

Output:



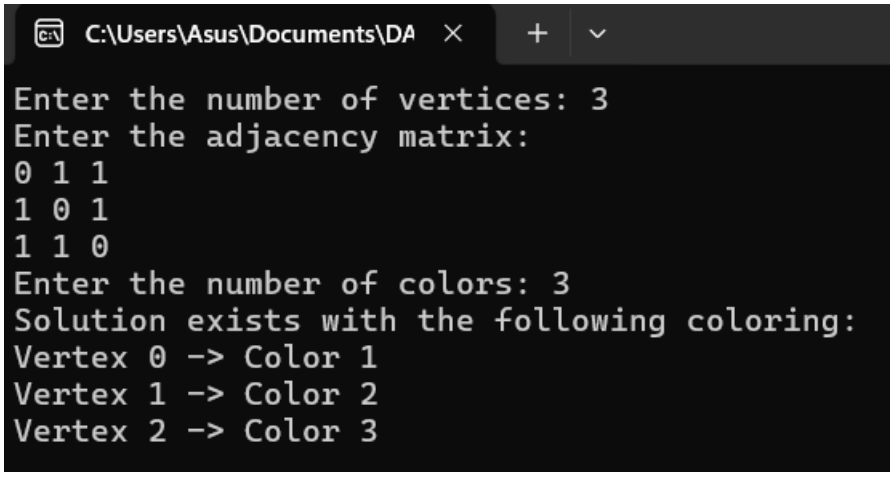
32. Graph coloring.

```
#include <stdio.h>
#define MAX 10

int graph[MAX][MAX], colors[MAX], n;

int isSafe(int node, int color) {
    for (int i = 0; i < n; i++) {
        if (graph[node][i] && colors[i] == color) {
            return 0;
        }
    }
    return 1;
```

```
}

int graphColoring(int node, int m) {
    if (node == n) {
        return 1;
    }
    for (int color = 1; color <= m; color++) {
        if (isSafe(node, color)) {
            colors[node] = color;
            if (graphColoring(node + 1, m)) {
                return 1;
            }
            colors[node] = 0;
        }
    }
    return 0;
}

int main() {
    int m;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    printf("Enter the adjacency matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }
    printf("Enter the number of colors: ");
    scanf("%d", &m);
```

```
    if (graphColoring(0, m)) {
        printf("Solution exists with the following coloring:\n");
        for (int i = 0; i < n; i++) {
            printf("Vertex %d -> Color %d\n", i, colors[i]);
        }
    } else {
        printf("No solution exists with %d colors.\n", m);
    }
    return 0;
}
```

Output:



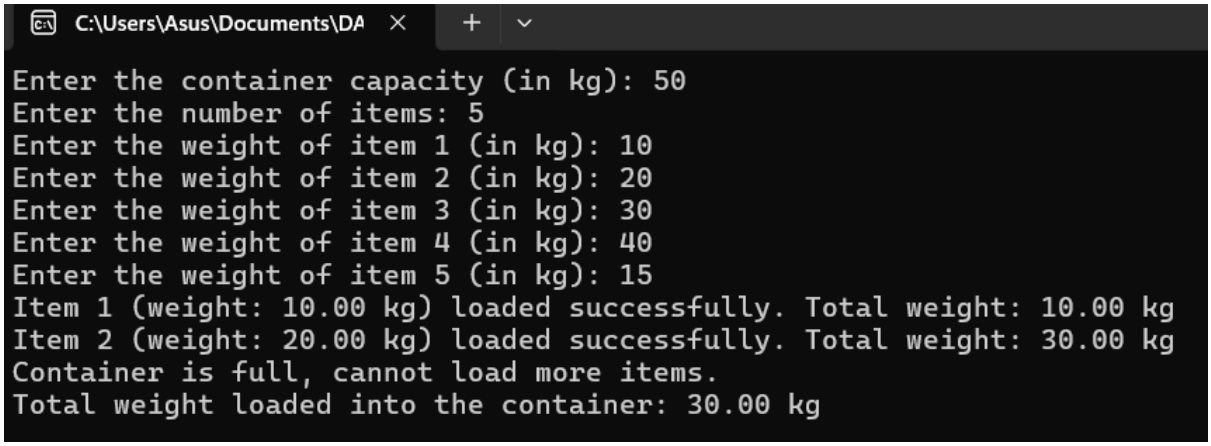
33. Container loader problem.

```
#include <stdio.h>

int main() {
    int numItems, i;
    float capacity, totalWeight = 0.0, itemWeight;

    // Get container capacity from user
    printf("Enter the container capacity (in kg): ");
```

```
    scanf("%f", &capacity);

    // Get the number of items to load
    printf("Enter the number of items: ");
    scanf("%d", &numItems);

    float itemWeights[numItems];

    // Get the weights of all items from user
    for(i = 0; i < numItems; i++) {
        printf("Enter the weight of item %d (in kg): ", i + 1);
        scanf("%f", &itemWeights[i]);
    }

    // Try to load items into the container
    for(i = 0; i < numItems; i++) {
        itemWeight = itemWeights[i];

        // Check if adding the item exceeds the container capacity
        if(totalWeight + itemWeight > capacity) {
            printf("Container is full, cannot load more items.\n");
            break;
        } else {
            totalWeight += itemWeight;
            printf("Item %d (weight: %.2f kg) loaded successfully. Total weight: %.2f
kg\n", i + 1, itemWeight, totalWeight);
        }
    }

    if(totalWeight == capacity) {
        printf("Container is now full.\n");
```

```
    } else {
        printf("Total weight loaded into the container: %.2f kg\n", totalWeight);
    }

    return 0;
}
```

Output:

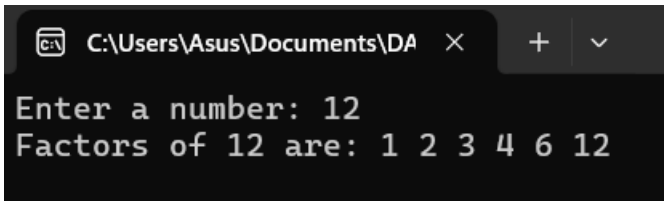


34. Factors of a given number.

```
#include <stdio.h>

int main() {
    int n, i;

    // Get the value of n from the user
    printf("Enter a number: ");
    scanf("%d", &n);

    printf("Factors of %d are: ", n);

    // Loop from 1 to n to find the factors
    for(i = 1; i <= n; i++) {
        if(n % i == 0) { // Check if i is a factor of n
```

```
            printf("%d ", i);
        }
    }

    printf("\n");
    return 0;
}
```

Output:



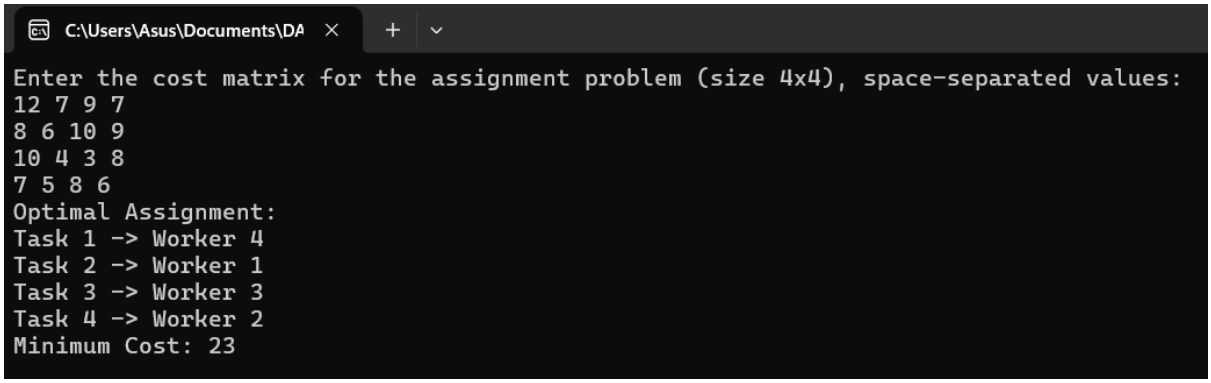
35. Assignment problem.

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

#define N 4  // Size of the matrix (number of tasks and workers)

int cost[N][N];
int finalAssignment[N];
int minCost = INT_MAX;

void printSolution(int assignment[], int costMatrix[N][N]) {
    printf("Optimal Assignment:\n");
    for (int i = 0; i < N; i++) {
        printf("Task %d -> Worker %d\n", i + 1, assignment[i] + 1);
    }
    printf("Minimum Cost: %d\n", minCost);
}
```

```
int calculateCost(int assignment[], int costMatrix[N][N]) {
    int totalCost = 0;
    for (int i = 0; i < N; i++) {
        totalCost += costMatrix[i][assignment[i]];
    }
    return totalCost;
}

void boundAndBranch(int costMatrix[N][N], int assignment[], int n, int level, int
currentCost, int visited[]) {
    if (level == n) {
        // Base case: all tasks assigned
        if (currentCost < minCost) {
            minCost = currentCost;
            for (int i = 0; i < n; i++) {
                finalAssignment[i] = assignment[i];
            }
        }
        return;
    }

    // Loop through all workers to try assigning tasks
    for (int i = 0; i < n; i++) {
        if (!visited[i]) {
            visited[i] = 1;
            assignment[level] = i;
            int newCost = currentCost + costMatrix[level][i];
            if (newCost < minCost) {
                boundAndBranch(costMatrix, assignment, n, level + 1, newCost, visited);
            }
            visited[i] = 0;  // Backtrack
```

```
        }
    }
}

int main() {
    int assignment[N] = {-1, -1, -1, -1};  // Holds the final assignment
    int visited[N] = {0, 0, 0, 0};  // Keeps track of assigned workers

    // Input the entire cost matrix at once
    printf("Enter the cost matrix for the assignment problem (size %dx%d), space-separated values:\n", N, N);
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            scanf("%d", &cost[i][j]);
        }
    }

    boundAndBranch(cost, assignment, N, 0, 0, visited);
    printSolution(finalAssignment, cost);

    return 0;
}
```

Output:

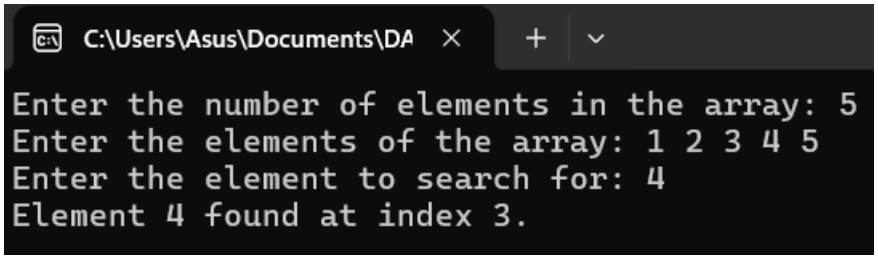


36. Linear search.

```
#include <stdio.h>

int linearSearch(int arr[], int size, int target) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == target) {
            return i;  // Return index if element is found
        }
    }
    return -1;  // Return -1 if element is not found
}

int main() {
    int size, target;

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

    int arr[size];  // Declare the array with the given size

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

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

```
    int result = linearSearch(arr, size, target);

    if (result != -1) {
        printf("Element %d found at index %d.\n", target, result);
    } else {
        printf("Element %d not found in the array.\n", target);
    }

    return 0;
}
```

Output:



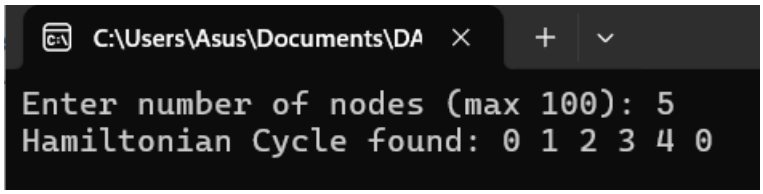
37. Hamiltonian circuit.

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_NODES 100
void generateCompleteGraph(int graph[MAX_NODES][MAX_NODES], int nodes) {
    for (int i = 0; i < nodes; i++)
        for (int j = 0; j < nodes; j++)
            graph[i][j] = (i != j);
}
bool isSafe(int v, int graph[MAX_NODES][MAX_NODES], int path[], int pos) {
    if (!graph[path[pos - 1]][v]) return false;
    for (int i = 0; i < pos; i++)
        if (path[i] == v) return false;
```

```
    return true;
}
bool hamiltonianCycleUtil(int graph[MAX_NODES][MAX_NODES], int path[], int pos,
int nodes) {
    if (pos == nodes) return graph[path[pos - 1]][path[0]];
    for (int v = 1; v < nodes; v++) {
        if (isSafe(v, graph, path, pos)) {
            path[pos] = v;
            if (hamiltonianCycleUtil(graph, path, pos + 1, nodes)) return true;
            path[pos] = -1;
        }
    }
    return false;
}
void findHamiltonianCycle(int graph[MAX_NODES][MAX_NODES], int nodes) {
    int path[MAX_NODES];
    for (int i = 0; i < nodes; i++) path[i] = -1;
    path[0] = 0;
    if (hamiltonianCycleUtil(graph, path, 1, nodes)) {
        printf("Hamiltonian Cycle found: ");
        for (int i = 0; i < nodes; i++) printf("%d ", path[i]);
        printf("%d\n", path[0]);
    } else {
        printf("No Hamiltonian Cycle found.\n");
    }
}
int main() {
    int nodes, graph[MAX_NODES][MAX_NODES];
    printf("Enter number of nodes (max %d): ", MAX_NODES);
    scanf("%d", &nodes);
    if (nodes < 3 || nodes > MAX_NODES) {
```

```
        printf("Invalid number of nodes.\n");
        return 1;
    }
    generateCompleteGraph(graph, nodes);
    findHamiltonianCycle(graph, nodes);
    return 0;
}
```

Output:



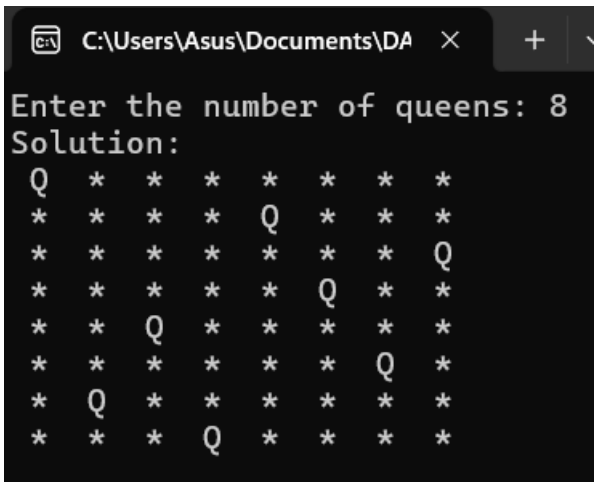
38. N queen's problem.

```
#include <stdio.h>
#include <stdlib.h>  // For abs()

// Function to print the board with queens placed on it
void printSolution(int board[], int N) {
    printf("Solution:\n");
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (board[i] == j) {
                printf(" Q ");  // Queen is placed at [i, j]
            } else {
                printf(" * ");  // Empty space
            }
        }
        printf("\n");
    }
    printf("\n");
```

```
}

// Function to check if it's safe to place a queen at [row, col]
int isSafe(int board[], int row, int col, int N) {
    for (int i = 0; i < row; i++) {
        if (board[i] == col || abs(board[i] - col) == abs(i - row)) {
            return 0;  // Conflict with another queen
        }
    }
    return 1;  // No conflict
}

// Backtracking function to solve the N-Queens problem
int solveNQueens(int board[], int row, int N) {
    if (row == N) {
        // All queens are placed successfully, print the solution
        printSolution(board, N);
        return 1;  // Return true after finding the first solution
    }

    for (int col = 0; col < N; col++) {
        if (isSafe(board, row, col, N)) {
            board[row] = col;  // Place queen at [row, col]
            if (solveNQueens(board, row + 1, N)) {
                return 1;  // Stop after finding the first solution
            }
        }
    }
    return 0;  // No solution found
}
```

```
int main() {
    int N;

    // Input the number of queens (size of the board)
    printf("Enter the number of queens: ");
    scanf("%d", &N);

    // Initialize the board
    int board[N];
    for (int i = 0; i < N; i++) {
        board[i] = -1;  // No queens placed initially
    }

    // Solve the N-Queens problem
    if (!solveNQueens(board, 0, N)) {
        printf("No solution exists for %d queens.\n", N);
    }

    return 0;
}
```

Output:



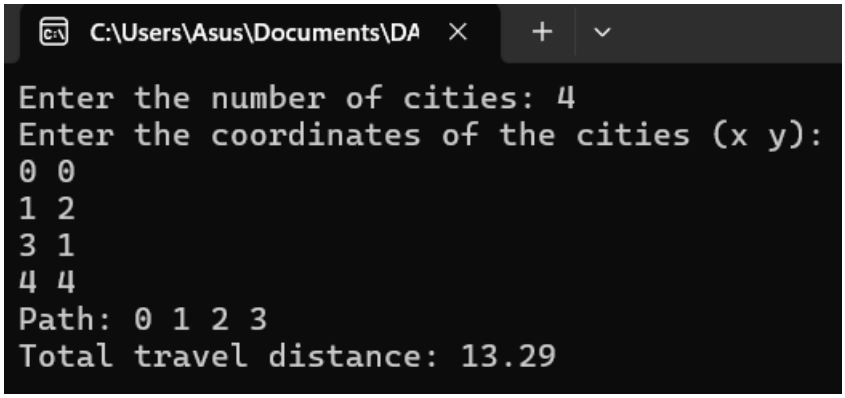
39. Approximation algorithm.

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

#define MAX_CITIES 10
#define INF 99999

// Function to calculate the distance between two cities (Euclidean distance)
double distance(int city1[], int city2[]) {
    return sqrt(pow(city1[0] - city2[0], 2) + pow(city1[1] - city2[1], 2));
}

// Nearest Neighbor Approximation Algorithm for TSP
double nearestNeighbor(int cities[][2], int n) {
    int visited[n];
    int path[n];
    double totalDistance = 0.0;

    // Initialize visited array
    for (int i = 0; i < n; i++) {
        visited[i] = 0;  // No cities are visited initially
    }

    visited[0] = 1;  // Start at the first city
    path[0] = 0;  // First city in the path
    int currentCity = 0;

    // Iterate through all cities
    for (int i = 1; i < n; i++) {
```

```
    double minDist = INF;
    int nextCity = -1;

    // Find the nearest unvisited city
    for (int j = 0; j < n; j++) {
        if (!visited[j]) {
            double dist = distance(cities[currentCity], cities[j]);
            if (dist < minDist) {
                minDist = dist;
                nextCity = j;
            }
        }
    }

    // Mark the next city as visited and add it to the path
    visited[nextCity] = 1;
    path[i] = nextCity;
    totalDistance += minDist;
    currentCity = nextCity;
}

// Add the distance to return to the start city
totalDistance += distance(cities[currentCity], cities[0]);

// Print the path
printf("Path: ");
for (int i = 0; i < n; i++) {
    printf("%d ", path[i]);
}
printf("\n");
```

```
    return totalDistance;
}

int main() {
    int n;

    // Input the number of cities
    printf("Enter the number of cities: ");
    scanf("%d", &n);

    // Input the coordinates of the cities
    int cities[n][2];
    printf("Enter the coordinates of the cities (x y):\n");
    for (int i = 0; i < n; i++) {
        scanf("%d %d", &cities[i][0], &cities[i][1]);
    }

    // Call the nearest neighbor algorithm
    double totalCost = nearestNeighbor(cities, n);

    printf("Total travel distance: %.2f\n", totalCost);

    return 0;
}
```

Output:



40. Min and max in array.

```
#include <stdio.h>

void findMinMax(int arr[], int n) {
    int min = arr[0];
    int max = arr[0];

    for (int i = 1; i < n; i++) {
        if (arr[i] < min) {
            min = arr[i];
        }
        if (arr[i] > max) {
            max = arr[i];
        }
    }

    printf("Minimum element: %d\n", min);
    printf("Maximum element: %d\n", max);
}

int main() {
    int n;
```

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

    int arr[n];

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

    // Call the function to find the min and max
    findMinMax(arr, n);

    return 0;
}
```

Output: