**List of array and matrix programming exercises**

1. Write a C program to read and print elements of array.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 1000 // Maximum array size

int main()

{

int arr[MAX\_SIZE]; // Declare an array of MAX\_SIZE

int i, N;

/\* Input array size \*/

printf("Enter size of array: ");

scanf("%d", &N);

/\* Input elements in array \*/

printf("Enter %d elements in the array : ", N);

for(i=0; i<N; i++)

{

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

}

/\*

\* Print all elements of array

\*/

printf("\nElements in array are: ");

for(i=0; i<N; i++)

{

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

}

return 0;

}

1. Write a C program to print all negative elements in an array.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum array size

int main()

{

int arr[MAX\_SIZE]; // Declare array of MAX\_SIZE

int i, N;

/\* Input size of the array \*/

printf("Enter size of the array : ");

scanf("%d", &N);

/\* Input elements in the array \*/

printf("Enter elements in array : ");

for(i=0; i<N; i++)

{

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

}

printf("\nAll negative elements in array are : ");

for(i=0; i<N; i++)

{

/\* If current array element is negative \*/

if(arr[i] < 0)

{

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

}

}

return 0;

}

1. Write a C program to find sum of all array elements.

Source Code

#include <stdio.h>

#define MAX\_SIZE 100

int main()

{

int arr[MAX\_SIZE];

int i, n, sum=0;

/\* Input size of the array \*/

printf("Enter size of the array: ");

scanf("%d", &n);

/\* Input elements in array \*/

printf("Enter %d elements in the array: ", n);

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

{

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

// Add each array element to sum

sum += arr[i];

}

printf("Sum of all elements of array = %d", sum);

return 0;

}

1. Write a C program to find maximum and minimum element in an array.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum array size

int main()

{

int arr[MAX\_SIZE];

int i, max, min, size;

/\* Input size of the array \*/

printf("Enter size of the array: ");

scanf("%d", &size);

/\* Input array elements \*/

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

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

{

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

}

/\* Assume first element as maximum and minimum \*/

max = arr[0];

min = arr[0];

/\*

\* Find maximum and minimum in all array elements.

\*/

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

{

/\* If current element is greater than max \*/

if(arr[i] > max)

{

max = arr[i];

}

/\* If current element is smaller than min \*/

if(arr[i] < min)

{

min = arr[i];

}

}

/\* Print maximum and minimum element \*/

printf("Maximum element = %d\n", max);

printf("Minimum element = %d", min);

return 0;

}

1. Write a C program to find second largest element in an array.

**Source Code**

#include <stdio.h>

#include <limits.h> // For INT\_MIN

#define MAX\_SIZE 1000 // Maximum array size

int main()

{

int arr[MAX\_SIZE], size, i;

int max1, max2;

/\* Input size of the array \*/

printf("Enter size of the array (1-1000): ");

scanf("%d", &size);

/\* Input array elements \*/

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

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

{

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

}

max1 = max2 = INT\_MIN;

/\*

\* Check for first largest and second

\*/

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

{

if(arr[i] > max1)

{

/\*

\* If current element of the array is first largest

\* then make current max as second max

\* and then max as current array element

\*/

max2 = max1;

max1 = arr[i];

}

else if(arr[i] > max2 && arr[i] < max1)

{

/\*

\* If current array element is less than first largest

\* but is greater than second largest then make it

\* second largest

\*/

max2 = arr[i];

}

}

printf("First largest = %d\n", max1);

printf("Second largest = %d", max2);

return 0;

}

1. Write a C program to count total number of even and odd elements in an array.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 100 //Maximum size of the array

int main()

{

int arr[MAX\_SIZE];

int i, size, even, odd;

/\* Input size of the array \*/

printf("Enter size of the array: ");

scanf("%d", &size);

/\* Input array elements \*/

printf("Enter %d elements in array: ", size);

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

{

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

}

/\* Assuming that there are 0 even and odd elements \*/

even = 0;

odd = 0;

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

{

/\* If the current element of array is even then increment even count \*/

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

{

even++;

}

else

{

odd++;

}

}

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

printf("Total odd elements: %d", odd);

return 0;

}

1. Write a C program to count total number of negative elements in an array.

**Soure Code**

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum array size

int main()

{

int arr[MAX\_SIZE]; // Declares array of size 100

int i, size, count = 0;

/\* Input size of array \*/

printf("Enter size of the array : ");

scanf("%d", &size);

/\* Input array elements \*/

printf("Enter elements in array : ");

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

{

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

}

/\*

\* Count total negative elements in array

\*/

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

{

/\* Increment count if current array element is negative \*/

if(arr[i] < 0)

{

count++;

}

}

printf("\nTotal negative elements in array = %d", count);

return 0;

}

1. Write a C program to copy all elements from an array to another array.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 100

int main()

{

int source[MAX\_SIZE], dest[MAX\_SIZE];

int i, size;

/\* Input size of the array \*/

printf("Enter the size of the array : ");

scanf("%d", &size);

/\* Input array elements \*/

printf("Enter elements of source array : ");

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

{

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

}

/\*

\* Copy all elements from source array to dest array

\*/

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

{

dest[i] = source[i];

}

/\*

\* Print all elements of source array

\*/

printf("\nElements of source array are : ");

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

{

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

}

/\*

\* Print all elements of dest array

\*/

printf("\nElements of dest array are : ");

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

{

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

}

return 0;

}

1. Write a C program to insert an element in an array.

Source Code

#include <stdio.h>

#define MAX\_SIZE 100

int main()

{

int arr[MAX\_SIZE];

int i, size, num, pos;

/\* Input size of the array \*/

printf("Enter size of the array : ");

scanf("%d", &size);

/\* Input elements in array \*/

printf("Enter elements in array : ");

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

{

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

}

/\* Input new element and position to insert \*/

printf("Enter element to insert : ");

scanf("%d", &num);

printf("Enter the element position : ");

scanf("%d", &pos);

/\* If position of element is not valid \*/

if(pos > size+1 || pos <= 0)

{

printf("Invalid position! Please enter position between 1 to %d", size);

}

else

{

/\* Make room for new array element by shifting to right \*/

for(i=size; i>=pos; i--)

{

arr[i] = arr[i-1];

}

/\* Insert new element at given position and increment size \*/

arr[pos-1] = num;

size++;

/\* Print array after insert operation \*/

printf("Array elements after insertion : ");

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

{

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

}

}

return 0;

}

1. Write a C program to delete an element from an array at specified position.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 100

int main()

{

int arr[MAX\_SIZE];

int i, size, pos;

/\* Input size and element in array \*/

printf("Enter size of the array : ");

scanf("%d", &size);

printf("Enter elements in array : ");

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

{

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

}

/\* Input element position to delete \*/

printf("Enter the element position to delete : ");

scanf("%d", &pos);

/\* Invalid delete position \*/

if(pos < 0 || pos > size)

{

printf("Invalid position! Please enter position between 1 to %d", size);

}

else

{

/\* Copy next element value to current element \*/

for(i=pos-1; i<size-1; i++)

{

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

}

/\* Decrement array size by 1 \*/

size--;

}

/\* Print array after deletion \*/

printf("\nElements of array after delete are : ");

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

{

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

}

return 0;

}

1. Write a C program to count frequency of each element in an array.

**Source Code**

#include <stdio.h>

int main()

{

int arr[100], freq[100];

int size, i, j, count;

/\* Input size of array \*/

printf("Enter size of array: ");

scanf("%d", &size);

/\* Input elements in array \*/

printf("Enter elements in array: ");

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

{

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

/\* Initially initialize frequencies to -1 \*/

freq[i] = -1;

}

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

{

count = 1;

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

{

/\* If duplicate element is found \*/

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

{

count++;

/\* Make sure not to count frequency of same element again \*/

freq[j] = 0;

}

}

/\* If frequency of current element is not counted \*/

if(freq[i] != 0)

{

freq[i] = count;

}

}

/\*

\* Print frequency of each element

\*/

printf("\nFrequency of all elements of array : \n");

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

{

if(freq[i] != 0)

{

printf("%d occurs %d times\n", arr[i], freq[i]);

}

}

return 0;

}

1. Write a C program to print all unique elements in the array.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 100

int main()

{

int arr[MAX\_SIZE], freq[MAX\_SIZE];

int size, i, j, count;

/\* Input size of array and elements in array \*/

printf("Enter size of array: ");

scanf("%d", &size);

printf("Enter elements in array: ");

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

{

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

freq[i] = -1;

}

/\* Find frequency of each element \*/

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

{

count = 1;

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

{

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

{

count++;

freq[j] = 0;

}

}

if(freq[i] != 0)

{

freq[i] = count;

}

}

/\* Print all unique elements of array \*/

printf("\nUnique elements in the array are: ");

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

{

if(freq[i] == 1)

{

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

}

}

return 0;

}

1. Write a C program to count total number of duplicate elements in an array.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum array size

int main()

{

int arr[MAX\_SIZE];

int i, j, size, count = 0;

/\* Input size of array \*/

printf("Enter size of the array : ");

scanf("%d", &size);

/\* Input elements in array \*/

printf("Enter elements in array : ");

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

{

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

}

/\*

\* Find all duplicate elements in array

\*/

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

{

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

{

/\* If duplicate found then increment count by 1 \*/

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

{

count++;

break;

}

}

}

printf("\nTotal number of duplicate elements found in array = %d", count);

return 0;

}

1. Write a C program to delete all duplicate elements from an array.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum size of the array

int main()

{

int arr[MAX\_SIZE]; // Declares an array of size 100

int size; // Total number of elements in array

int i, j, k; // Loop control variables

/\* Input size of the array \*/

printf("Enter size of the array : ");

scanf("%d", &size);

/\* Input elements in the array \*/

printf("Enter elements in array : ");

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

{

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

}

/\*

\* Find duplicate elements in array

\*/

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

{

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

{

/\* If any duplicate found \*/

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

{

/\* Delete the current duplicate element \*/

for(k=j; k<size; k++)

{

arr[k] = arr[k + 1];

}

/\* Decrement size after removing duplicate element \*/

size--;

/\* If shifting of elements occur then don't increment j \*/

j--;

}

}

}

/\*

\* Print array after deleting duplicate elements

\*/

printf("\nArray elements after deleting duplicates : ");

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

{

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

}

return 0;

}

1. Write a C program to merge two array to third array.

**Souce Code**

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum size of the array

int main()

{

int arr1[MAX\_SIZE], arr2[MAX\_SIZE], mergeArray[MAX\_SIZE \* 2];

int size1, size2, mergeSize;

int index1, index2, mergeIndex;

int i;

/\* Input size of first array \*/

printf("Enter the size of first array : ");

scanf("%d", &size1);

/\* Input elements in first array \*/

printf("Enter elements in first array : ");

for(i=0; i<size1; i++)

{

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

}

/\* Input size of second array \*/

printf("\nEnter the size of second array : ");

scanf("%d", &size2);

/\* Input elements in second array \*/

printf("Enter elements in second array : ");

for(i=0; i<size2; i++)

{

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

}

mergeSize = size1 + size2;

/\*

\* Merge two array in ascending order

\*/

index1 = 0;

index2 = 0;

for(mergeIndex=0; mergeIndex < mergeSize; mergeIndex++)

{

/\*

\* If all elements of one array

\* is merged to final array

\*/

if(index1 >= size1 || index2 >= size2)

{

break;

}

if(arr1[index1] < arr2[index2])

{

mergeArray[mergeIndex] = arr1[index1];

index1++;

}

else

{

mergeArray[mergeIndex] = arr2[index2];

index2++;

}

}

/\*

\* Merge remaining array elements

\*/

while(index1 < size1)

{

mergeArray[mergeIndex] = arr1[index1];

mergeIndex++;

index1++;

}

while(index2 < size2)

{

mergeArray[mergeIndex] = arr2[index2];

mergeIndex++;

index2++;

}

/\*

\* Print merged array

\*/

printf("\nArray merged in ascending order : ");

for(i=0; i<mergeSize; i++)

{

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

}

return 0;

}

1. Write a C program to find reverse of an array.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 100 // Defines maximum size of array

int main()

{

int arr[MAX\_SIZE];

int size, i;

/\* Input size of array \*/

printf("Enter size of the array: ");

scanf("%d", &size);

/\* Input array elements \*/

printf("Enter elements in array: ");

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

{

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

}

/\*

\* Print array in reversed order

\*/

printf("\nArray in reverse order: ");

for(i = size-1; i>=0; i--)

{

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

}

return 0;

}

1. Write a C program to put even and odd elements of array in two separate array.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 1000 // Maximum size of the array

/\* Function to print array \*/

void printArray(int arr[], int len);

int main()

{

int arr[MAX\_SIZE];

int even[MAX\_SIZE], odd[MAX\_SIZE];

int evenCount, oddCount;

int i, size;

/\* Input size of the array \*/

printf("Enter size of the array: ");

scanf("%d", &size);

/\* Input elements in array \*/

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

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

{

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

}

evenCount = 0;

oddCount = 0;

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

{

// If arr[i] is odd

// Binary AND operator copies a bit to the resul if it exists in both operands.

if(arr[i] & 1)

{

odd[oddCount] = arr[i];

oddCount++;

}

else

{

even[evenCount] = arr[i];

evenCount++;

}

}

printf("\nElements of even array: \n");

printArray(even, evenCount);

printf("\nElements of odd array: \n");

printArray(odd, oddCount);

return 0;

}

/\*\*

\* Print the entire integer array

\* @arr Integer array to be displayed or printed on screen

\* @len Length of the array

\*/

void printArray(int arr[], int len)

{

int i;

printf("Elements in the array: ");

for(i=0; i<len; i++)

{

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

}

printf("\n");

}

1. Write a C program to search an element in an array.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum array size

int main()

{

int arr[MAX\_SIZE];

int size, i, toSearch, found;

/\* Input size of array \*/

printf("Enter size of array: ");

scanf("%d", &size);

/\* Input elements of array \*/

printf("Enter elements in array: ");

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

{

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

}

printf("\nEnter element to search: ");

scanf("%d", &toSearch);

/\* Assume that element does not exists in array \*/

found = 0;

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

{

/\*

\* If element is found in array then raise found flag

\* and terminate from loop.

\*/

if(arr[i] == toSearch)

{

found = 1;

break;

}

}

/\*

\* If element is not found in array

\*/

if(found == 1)

{

printf("\n%d is found at position %d", toSearch, i + 1);

}

else

{

printf("\n%d is not found in the array", toSearch);

}

return 0;

}

1. Write a C program to sort array elements in ascending or descending order.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum array size

int main()

{

int arr[MAX\_SIZE];

int size;

int i, j, temp;

/\* Input size of array \*/

printf("Enter size of array: ");

scanf("%d", &size);

/\* Input elements in array \*/

printf("Enter elements in array: ");

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

{

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

}

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

{

/\*

\* Place currently selected element array[i]

\* to its correct place.

\*/

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

{

/\*

\* Swap if currently selected array element

\* is not at its correct position.

\*/

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

{

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

/\* Print the sorted array \*/

printf("\nElements of array in ascending order: ");

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

{

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

}

return 0;

}

1. Write a C program to sort even and odd elements of array separately.

**Source Code**

#include <stdio.h>

#include <limits.h> //Used for INT\_MAX

#define MAX\_SIZE 1000 //Maximum size of the array

/\*

\* Functions used in this program

\*/

void arrange(int arr[], int len, int pivot);

void sort(int arr[], int start, int end);

void print(int arr[], int len);

int main()

{

int arr[MAX\_SIZE], i, n;

int pivot, evenCount, oddCount;

pivot = 0;

evenCount = oddCount = 0;

/\*

\* Reads size and elements in the array

\*/

printf("Enter size of the array: ");

scanf("%d", &n);

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

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

{

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

// If current element is odd then increase pivot

if(arr[i] & 1)

oddCount++;

else

evenCount++;

}

/\*

\* Pivot is position that separates even and odd elements

\*/

pivot = (evenCount > oddCount) ? evenCount : oddCount;

print(arr, n);

// Arranges all even and odd elements sequentially

arrange(arr, n, pivot);

// Print elements after arranging even and odd elements

printf("\nElements after arranging even and odd elements separately\n");

print(arr, n);

//Sorts even part of the array

sort(arr, pivot, n);

//Sorts odd part of the array

sort(arr, 0, pivot);

//Prints the final sorted array

printf("\nFinal array after sorting even and odd elements separately\n");

print(arr, n);

return 0;

}

/\*\*

\* Arranges all even and odd elements of the array separately. Puts

\* all even elements first then all odd elements.

\*/

void arrange(int arr[], int len, int pivot)

{

int i, j, temp;

for(i=0; i<pivot; i++)

{

/\*

\* If current element of array is odd put it into

\* odd element place

\*/

if(arr[i] & 1)

{

for(j=pivot; j<len; j++)

{

//Look for an even element then swap with odd element

if(!(arr[j] & 1))

{

temp = arr[j];

arr[j] = arr[i];

arr[i] = temp;

break;

}

}

}

}

}

/\*\*

\* Sorts the elements of array within a range

\*/

void sort(int arr[], int start, int end)

{

int i, j, temp;

int len = start + end;

for(i=start; i<len; i++)

{

for(j=i+1; j<len; j++)

{

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

{

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

}

/\*\*

\* Prints the entire integer array

\*/

void print(int arr[], int len)

{

int i;

printf("Elements in the array: ");

for(i=0; i<len; i++)

{

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

}

printf("\n");

}

1. Write a C program to left rotate an array.

**Source Code**

#include <stdio.h>

#define SIZE 10 /\* Size of the array \*/

void printArray(int arr[]);

void rotateByOne(int arr[]);

int main()

{

int i, N;

int arr[SIZE];

printf("Enter 10 elements array: ");

for(i=0; i<SIZE; i++)

{

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

}

printf("Enter number of times to left rotate: ");

scanf("%d", &N);

/\* Actual rotation \*/

N = N % SIZE;

/\* Print array before rotation \*/

printf("Array before rotationn");

printArray(arr);

/\* Rotate array n times \*/

for(i=1; i<=N; i++)

{

rotateByOne(arr);

}

/\* Print array after rotation \*/

printf("\n\nArray after rotation\n");

printArray(arr);

return 0;

}

void rotateByOne(int arr[])

{

int i, first;

/\* Store first element of array \*/

first = arr[0];

for(i=0; i<SIZE-1; i++)

{

/\* Move each array element to its left \*/

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

}

/\* Copies the first element of array to last \*/

arr[SIZE-1] = first;

}

/\*\*

\* Print the given array

\*/

void printArray(int arr[])

{

int i;

for(i=0; i<SIZE; i++)

{

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

}

}

1. Write a C program to right rotate an array.

**Source Code**

#include <stdio.h>

#define SIZE 10 /\* Size of the array \*/

void printArray(int arr[]);

void rotateByOne(int arr[]);

int main()

{

int i, N;

int arr[SIZE];

printf("Enter 10 elements array: ");

for(i=0; i<SIZE; i++)

{

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

}

printf("Enter number of times to right rotate: ");

scanf("%d", &N);

/\* Actual rotation \*/

N = N % SIZE;

/\* Print array before rotation \*/

printf("Array before rotationn");

printArray(arr);

/\* Rotate array n times \*/

for(i=1; i<=N; i++)

{

rotateByOne(arr);

}

/\* Print array after rotation \*/

printf("\n\nArray after rotation\n");

printArray(arr);

return 0;

}

void rotateByOne(int arr[])

{

int i, last;

/\* Store last element of array \*/

last = arr[SIZE - 1];

for(i=SIZE-1; i>0; i--)

{

/\* Move each array element to its right \*/

arr[i] = arr[i - 1];

}

/\* Copy last element of array to first \*/

arr[0] = last;

}

/\*\*

\* Print the given array

\*/

void printArray(int arr[])

{

int i;

for(i=0; i<SIZE; i++)

{

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

}

}

**List of matrix programming exercises**

1. Write a C program to add two matrices.

**Source Code**

#include <stdio.h>

#define SIZE 3 // Size of the matrix

int main()

{

int A[SIZE][SIZE]; // Matrix 1

int B[SIZE][SIZE]; // Matrix 2

int C[SIZE][SIZE]; // Resultant matrix

int row, col;

/\* Input elements in first matrix\*/

printf("Enter elements in matrix A of size 3x3: \n");

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

scanf("%d", &A[row][col]);

}

}

/\* Input elements in second matrix \*/

printf("\nEnter elements in matrix B of size 3x3: \n");

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

scanf("%d", &B[row][col]);

}

}

/\*

\* Add both matrices A and B entry wise or element wise

\* and stores result in matrix C

\*/

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

/\* Cij = Aij + Bij \*/

C[row][col] = A[row][col] + B[row][col];

}

}

/\* Print the value of resultant matrix C \*/

printf("\nSum of matrices A+B = \n");

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

printf("%d ", C[row][col]);

}

printf("\n");

}

return 0;

}

1. Write a C program to subtract two matrices.

**Source Code**

#include <stdio.h>

#define SIZE 3 // Size of the matrix

int main()

{

int A[SIZE][SIZE]; // Matrix 1

int B[SIZE][SIZE]; // Matrix 2

int C[SIZE][SIZE]; // Resultant matrix

int row, col;

/\* Input elements in first matrix\*/

printf("Enter elements in matrix A of size 3x3: \n");

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

scanf("%d", &A[row][col]);

}

}

/\* Input elements in second matrix \*/

printf("\nEnter elements in matrix B of size 3x3: \n");

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

scanf("%d", &B[row][col]);

}

}

/\*

\* Add both matrices A and B entry wise or element wise

\* and stores result in matrix C

\*/

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

/\* Cij = Aij - Bij \*/

C[row][col] = A[row][col] - B[row][col];

}

}

/\* Print the value of resultant matrix C \*/

printf("\nSum of matrices A+B = \n");

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

printf("%d ", C[row][col]);

}

printf("\n");

}

return 0;

}

1. Write a C program to perform Scalar matrix multiplication.

**Source Code**

#include <stdio.h>

#define SIZE 3 // Maximum size of the array

int main()

{

int A[SIZE][SIZE];

int num, row, col;

/\* Input elements in matrix from user \*/

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

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

scanf("%d", &A[row][col]);

}

}

/\* Input multiplier from user \*/

printf("Enter any number to multiply with matrix A: ");

scanf("%d", &num);

/\* Perform scalar multiplication of matrix \*/

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

/\* (cAij) = c . Aij \*/

A[row][col] = num \* A[row][col];

}

}

/\* Print result of scalar multiplication of matrix \*/

printf("\nResultant matrix c.A = \n");

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

printf("%d ", A[row][col]);

}

printf("\n");

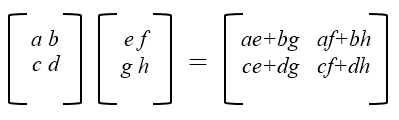
}

return 0;

}

1. Write a C program to multiply two matrices.

**Source Code**



#include <stdio.h>

#define SIZE 3 // Size of the matrix

int main()

{

int A[SIZE][SIZE]; // Matrix 1

int B[SIZE][SIZE]; // Matrix 2

int C[SIZE][SIZE]; // Resultant matrix

int row, col, i, sum;

/\* Input elements in first matrix from user \*/

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

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

scanf("%d", &A[row][col]);

}

}

/\* Input elements in second matrix from user \*/

printf("\nEnter elements in matrix B of size %dx%d: \n", SIZE, SIZE);

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

scanf("%d", &B[row][col]);

}

}

/\*

\* Multiply both matrices A\*B

\*/

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

sum = 0;

/\*

\* Multiply row of first matrix to column of second matrix

\* and store sum of product of elements in sum.

\*/

for(i=0; i<SIZE; i++)

{

sum += A[row][i] \* B[i][col];

}

C[row][col] = sum;

}

}

/\* Print product of the matrices \*/

printf("\nProduct of matrix A \* B = \n");

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

printf("%d ", C[row][col]);

}

printf("\n");

}

return 0;

}

1. Write a C program to check whether two matrices are equal or not.

**Source Code**

#include <stdio.h>

#define SIZE 3 // Matrix size

int main()

{

int A[SIZE][SIZE];

int B[SIZE][SIZE];

int row, col, isEqual;

/\* Input elements in first matrix from user \*/

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

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

scanf("%d", &A[row][col]);

}

}

/\* Input elements in second matrix from user \*/

printf("\nEnter elements in matrix B of size %dx%d: \n");

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

scanf("%d", &B[row][col]);

}

}

/\* Assumes that the matrices are equal \*/

isEqual = 1;

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

/\*

\* If the corresponding entries of matrices are not equal

\*/

if(A[row][col] != B[row][col])

{

isEqual = 0;

break;

}

}

}

/\*

\* Checks the value of isEqual

\* As per our assumption if isEqual contains 1 means both are equal

\* If it contains 0 means both are not equal

\*/

if(isEqual == 1)

{

printf("\nMatrix A is equal to Matrix B");

}

else

{

printf("\nMatrix A is not equal to Matrix B");

}

return 0;

}

1. Write a C program to find sum of main diagonal elements of a matrix.

**Source Code**

#include <stdio.h>

#define SIZE 3 // Matrix size

int main()

{

int A[SIZE][SIZE];

int row, col, sum = 0;

/\* Input elements in matrix from user \*/

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

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

scanf("%d", &A[row][col]);

}

}

/\* Find sum of main diagonal elements \*/

for(row=0; row<SIZE; row++)

{

sum = sum + A[row][row];

}

printf("\nSum of main diagonal elements = %d", sum);

return 0;

}

1. Write a C program to find sum of minor diagonal elements of a matrix.

**Source Code**

#include <stdio.h>

#define SIZE 3 // Matrix size

int main()

{

int A[SIZE][SIZE];

int row, col, sum = 0;

/\* Input elements in matrix from user \*/

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

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

scanf("%d", &A[row][col]);

}

}

/\* Find sum of minor diagonal elements \*/

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

/\*

\* If it is minor diagonal of matrix

\* Minor diagonal: i+j == N + 1

\* Since array elements starts from 0 hence i+j == (N + 1)-2

\*/

if(row+col == ((SIZE+1)-2))

{

sum += A[row][col];

}

}

}

printf("\nSum of minor diagonal elements = %d", sum);

return 0;

}

1. Write a C program to find sum of each row and column of a matrix.

**Source Code**

#include <stdio.h>

#define SIZE 3 // Matrix size

int main()

{

int A[SIZE][SIZE];

int row, col, sum = 0;

/\* Input elements in matrix from user \*/

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

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

scanf("%d", &A[row][col]);

}

}

/\* Calculate sum of elements of each row of matrix \*/

for(row=0; row<SIZE; row++)

{

sum = 0;

for(col=0; col<SIZE; col++)

{

sum += A[row][col];

}

printf("Sum of elements of Row %d = %d\n", row+1, sum);

}

/\* Find sum of elements of each columns of matrix \*/

for(row=0; row<SIZE; row++)

{

sum = 0;

for(col=0; col<SIZE; col++)

{

sum += A[col][row];

}

printf("Sum of elements of Column %d = %d\n", row+1, sum);

}

return 0;

}

1. Write a C program to interchange diagonals of a matrix.

**Source Code**

#include <stdio.h>

#define MAX\_ROWS 3

#define MAX\_COLS 3

int main()

{

int A[MAX\_ROWS][MAX\_COLS];

int row, col, size, temp;

/\* Input elements in matrix from user \*/

printf("Enter elements in matrix of size %dx%d: \n", MAX\_ROWS, MAX\_COLS);

for(row=0; row<MAX\_ROWS; row++)

{

for(col=0; col<MAX\_COLS; col++)

{

scanf("%d", &A[row][col]);

}

}

size = (MAX\_ROWS < MAX\_COLS) ? MAX\_ROWS : MAX\_COLS;

/\*

\* Interchange diagonal of the matrix

\*/

for(row=0; row<size; row++)

{

col = row;

temp = A[row][col];

A[row][col] = A[row][(size-col) - 1];

A[row][(size-col) - 1] = temp;

}

/\*

\* Print the interchanged diagonals matrix

\*/

printf("\nMatrix after diagonals interchanged: \n");

for(row=0; row<MAX\_ROWS; row++)

{

for(col=0; col<MAX\_COLS; col++)

{

printf("%d ", A[row][col]);

}

printf("\n");

}

return 0;

}

1. Write a C program to find upper triangular matrix.

**Source Code**

#include <stdio.h>

#define MAX\_ROWS 3

#define MAX\_COLS 3

int main()

{

int array[MAX\_ROWS][MAX\_COLS];

int row, col, isUpper;

/\* Input elements in matrix from user \*/

printf("Enter elements in matrix of size %dx%d: \n", MAX\_ROWS, MAX\_COLS);

for(row=0; row<MAX\_ROWS; row++)

{

for(col=0; col<MAX\_COLS; col++)

{

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

}

}

/\* Check Upper triangular matrix condition \*/

isUpper = 1;

for(row=0; row<MAX\_ROWS; row++)

{

for(col=0; col<MAX\_COLS; col++)

{

/\*

\* If elements below the main diagonal (col<row)

\* is not equal to zero then it is not upper triangular matrix

\*/

if(col<row && array[row][col]!=0)

{

isUpper = 0;

}

}

}

/\* Print elements of upper triangular matrix \*/

if(isUpper == 1)

{

printf("\nThe matrix is Upper triangular matrix.\n");

for(row=0; row<MAX\_ROWS; row++)

{

for(col=0; col<MAX\_COLS; col++)

{

printf("%d ", array[row][col]);

}

printf("\n");

}

}

else

{

printf("\nThe matrix is not Upper triangular matrix.");

}

return 0;

}

1. Write a C program to find lower triangular matrix.

**Source Code**

#include <stdio.h>

#define MAX\_ROWS 3

#define MAX\_COLS 3

int main()

{

int array[MAX\_ROWS][MAX\_COLS];

int row, col, isLower;

/\* Input elements in matrix from user \*/

printf("Enter elements in matrix of size %dx%d: \n", MAX\_ROWS, MAX\_COLS);

for(row=0; row<MAX\_ROWS; row++)

{

for(col=0; col<MAX\_COLS; col++)

{

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

}

}

/\* Check whether the matrix is lower triangular matrix \*/

isLower = 1;

for(row=0; row<MAX\_ROWS; row++)

{

for(col=0; col<MAX\_COLS; col++)

{

/\*

\* If elements above main diagonal(col>row)

\* is not equal to zero(array[row][col]!=0)

\*/

if(col>row && array[row][col]!=0)

{

isLower = 0;

}

}

}

/\*

\* If matrix is lower triangular matrix

\*/

if(isLower == 1)

{

printf("\nMatrix is Lower triangular matrix: \n");

/\* Print elements of lower triangular matrix \*/

for(row=0; row<MAX\_ROWS; row++)

{

for(col=0; col<MAX\_COLS; col++)

{

printf("%d ", array[row][col]);

}

printf("\n");

}

}

else

{

printf("\nMatrix is not a Lower triangular matrix");

}

return 0;

}

1. Write a C program to find sum of upper triangular matrix.

**Source Code**

#include <stdio.h>

#define MAX\_ROWS 3

#define MAX\_COLS 3

int main()

{

int A[MAX\_ROWS][MAX\_ROWS];

int row, col, sum = 0;

/\* Input elements in matrix from user \*/

printf("Enter elements in matrix of size %dx%d: \n", MAX\_ROWS, MAX\_COLS);

for(row=0; row<MAX\_ROWS; row++)

{

for(col=0; col<MAX\_COLS; col++)

{

scanf("%d", &A[row][col]);

}

}

/\* Find sum of upper triangular matrix \*/

for(row=0; row<MAX\_ROWS; row++)

{

for(col=0; col<MAX\_COLS; col++)

{

if(col>row)

{

sum += A[row][col];

}

}

}

printf("Sum of upper triangular matrix = %d", sum);

return 0;

}

1. Write a C program to find sum of lower triangular matrix.

**Source Code**

#include <stdio.h>

#define MAX\_ROWS 3

#define MAX\_COLS 3

int main()

{

int A[MAX\_ROWS][MAX\_COLS];

int row, col, sum = 0;

/\* Input elements in matrix from user \*/

printf("Enter elements in matrix of size %dx%d: \n", MAX\_ROWS, MAX\_COLS);

for(row=0; row<MAX\_ROWS; row++)

{

for(col=0; col<MAX\_COLS; col++)

{

scanf("%d", &A[row][col]);

}

}

/\* Find sum of lower triangular matrix \*/

for(row=0; row<MAX\_ROWS; row++)

{

for(col=0; col<MAX\_COLS; col++)

{

if(col<row)

{

sum += A[row][col];

}

}

}

printf("Sum of lower triangular matrix = %d", sum);

return 0;

}

1. Write a C program to find transpose of a matrix.

**Source Code**

#include <stdio.h>

#define MAX\_ROWS 3

#define MAX\_COLS 3

int main()

{

int A[MAX\_ROWS][MAX\_COLS]; // Original matrix

int B[MAX\_COLS][MAX\_ROWS]; // Transpose matrix

int row, col;

/\* Input elements in matrix A from user \*/

printf("Enter elements in matrix of size %dx%d: \n", MAX\_ROWS, MAX\_COLS);

for(row=0; row<MAX\_ROWS; row++)

{

for(col=0; col<MAX\_COLS; col++)

{

scanf("%d", &A[row][col]);

}

}

/\*

\* Find transpose of matrix A

\*/

for(row=0; row<MAX\_ROWS; row++)

{

for(col=0; col<MAX\_COLS; col++)

{

/\* Store each row of matrix A to each column of matrix B \*/

B[col][row] = A[row][col];

}

}

/\* Print the original matrix A \*/

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

for(row=0; row<MAX\_ROWS; row++)

{

for(col=0; col<MAX\_COLS; col++)

{

printf("%d ", A[row][col]);

}

printf("\n");

}

/\* Print the transpose of matrix A \*/

printf("Transpose of matrix A: \n");

for(row=0; row<MAX\_COLS; row++)

{

for(col=0; col<MAX\_ROWS; col++)

{

printf("%d ", B[row][col]);

}

printf("\n");

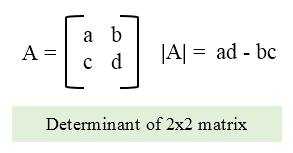
}

return 0;

}

1. Write a C program to find determinant of a matrix.

**Source Code**



#include <stdio.h>

#define SIZE 2 // Matrix size

int main()

{

int A[SIZE][SIZE];

int row, col;

long det;

/\* Input elements in matrix A from user \*/

printf("Enter elements in matrix of size 2x2: \n");

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

scanf("%d", &A[row][col]);

}

}

/\*

\* det(A) = ad - bc

\* a = A[0][0], b = A[0][1], c = A[1][0], d = A[1][1]

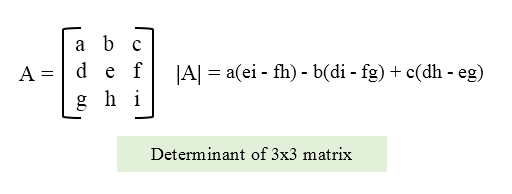
\*/

det = (A[0][0] \* A[1][1]) - (A[0][1] \* A[1][0]);

printf("Determinant of matrix A = %ld", det);

return 0;

}



#include <stdio.h>

#define SIZE 3 // Matrix size

int main()

{

int A[SIZE][SIZE];

int row, col;

int a, b, c, d, e, f, g, h, i;

long det;

/\* Input elements in matrix A from user \*/

printf("Enter elements in matrix of size 3x3: \n");

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

scanf("%d", &A[row][col]);

}

}

/\*

\* Used as a temporary variables to make calculation easy

\* | |

\* | a b c |

\* | d e f |

\* | g h i |

\* | |

\*/

a = A[0][0];

b = A[0][1];

c = A[0][2];

d = A[1][0];

e = A[1][1];

f = A[1][2];

g = A[2][0];

h = A[2][1];

i = A[2][2];

/\*

\* det(A) = a(ei - fh) - b(di - fg) + c(dh - eg)

\*/

det = (a\*(e\*i - f\*h)) - (b\*(d\*i - f\*g)) + (c\*(d\*h - e\*g));

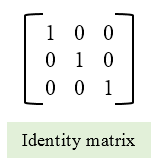
printf("Determinant of matrix A = %ld", det);

return 0;

}

1. Write a C program to check Identity matrix.

**Source Code**

 Identity matrix is a special square matrix whose [main diagonal](https://codeforwin.org/2015/07/c-program-to-find-sum-of-main-diagonal-elements-of-matrix.html) elements is equal to 1 and other elements are 0. Identity matrix is also known as unit matrix.

#include <stdio.h>

#define SIZE 3 // Matrix size

int main()

{

int A[SIZE][SIZE];

int row, col, isIdentity;

/\* Input elements in matrix from user \*/

printf("Enter elements in matrix of size 3x3: \n");

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

scanf("%d", &A[row][col]);

}

}

/\* Check whether it is Identity matrix or not \*/

isIdentity = 1;

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

if(row==col && A[row][col]!=1)

{

/\* If elements of main diagonal is not equal to 1 \*/

isIdentity = 0;

}

else if(row!=col && A[row][col]!=0)

{

/\* If other elements than main diagonal is not equal to 0 \*/

isIdentity = 0;

}

}

}

/\* If it is an Identity matrix \*/

if(isIdentity == 1)

{

printf("\nThe given matrix is an Identity Matrix.\n");

/\*

\* Print the Identity matrix

\*/

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

printf("%d ", A[row][col]);

}

printf("\n");

}

}

else

{

printf("The given matrix is not Identity Matrix");

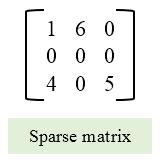
}

return 0;

}

1. Write a C program to check Sparse matrix.

**Source Code**



Sparse matrix is a special matrix with most of its elements are zero. We can also assume that if (m \* n) / 2 elements are zero then it is a sparse matrix.

#include <stdio.h>

#define SIZE 3

int main()

{

int A[SIZE][SIZE];

int row, col, total=0;

/\* Input elements in matrix from user \*/

printf("Enter elements in matrix of size 3x3: \n");

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

scanf("%d", &A[row][col]);

}

}

/\* Count total number of zero elements in the matrix \*/

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

/\* If the current element is zero \*/

if(A[row][col] == 0)

{

total++;

}

}

}

if(total >= (row \* col)/2)

{

printf("\nThe given matrix is a Sparse matrix.");

}

else

{

printf("\nThe given matrix is not Sparse matrix.");

}

return 0;

}

1. Write a C program to check Symmetric matrix.

**Source Code**

#include <stdio.h>

#define SIZE 3

int main()

{

int A[SIZE][SIZE]; // Original matrix

int B[SIZE][SIZE]; // Transpose matrix

int row, col, isSymmetric;

/\* Input elements in matrix A from user \*/

printf("Enter elements in matrix of size 3x3: \n");

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

scanf("%d", &A[row][col]);

}

}

/\*

\* Find transpose of matrix A

\*/

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

/\* Store each row of matrix A to each column of matrix B \*/

B[row][col] = A[col][row];

}

}

/\*

\* Check whether matrix A is equal to its transpose or not

\*/

isSymmetric = 1;

for(row=0; row<SIZE && isSymmetric; row++)

{

for(col=0; col<SIZE; col++)

{

/\* If matrix A is not equal to its transpose \*/

if(A[row][col] != B[row][col])

{

isSymmetric = 0;

break;

}

}

}

/\*

\* If the given matrix is symmetric.

\*/

if(isSymmetric == 1)

{

printf("\nThe given matrix is Symmetric matrix: \n");

for(row=0; row<SIZE; row++)

{

for(col=0; col<SIZE; col++)

{

printf("%d ", A[row][col]);

}

printf("\n");

}

}

else

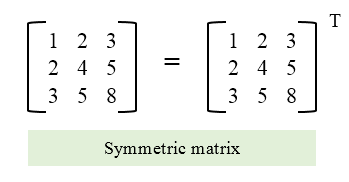
{

printf("\nThe given matrix is not Symmetric matrix.");

}

return 0;

}



Symmetric matrix is a square matrix which is equal to its transpose. A symmetric matrix is always a square matrix. Symmetric matrix **A** is defined as - **A** = **A**T

**List of string programming exercises**

1. Write a C program to find length of a string.

**Source Code**

#include <stdio.h>

#include <string.h>

#define MAX\_SIZE 100 // Maximum size of string

int main()

{

char text[MAX\_SIZE]; /\* Declares a string of size 100 \*/

int length;

printf("Enter any string: ");

gets(text);

/\* Call strlen() function to count length of string \*/

length = strlen(text);

printf("Length of '%s' = %d", text, length);

return 0;

}

1. Write a C program to copy one string to another string.

**Source Code**

#include <stdio.h>

#include <string.h>

#define MAX\_SIZE 100 // Maximum size of string

int main()

{

char text1[MAX\_SIZE], text2[MAX\_SIZE];

/\* Input original string from user \*/

printf("Enter any string: ");

gets(text1);

/\* Copy text1 to text2 using strcpy() \*/

strcpy(text2, text1);

printf("First string = %s\n", text1);

printf("Second string = %s\n", text2);

return 0;

}

1. Write a C program to concatenate two strings.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum string size

int main()

{

char str1[MAX\_SIZE], str2[MAX\_SIZE];

int i, j;

/\* Input two strings from user \*/

printf("Enter first string: ");

gets(str1);

printf("Enter second string: ");

gets(str2);

/\* Move till the end of str1 \*/

i=0;

while(str1[i] != '\0')

{

i++;

}

/\* Copy str2 to str1 \*/

j = 0;

while(str2[j] != '\0')

{

str1[i] = str2[j];

i++;

j++;

}

// Make sure that str1 is NULL terminated

str1[i] = '\0';

printf("Concatenated string = %s", str1);

return 0;

}

1. Write a C program to compare two strings.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum string size

/\* Compare function declaration \*/

int compare(char \* str1, char \* str2);

int main()

{

char str1[MAX\_SIZE], str2[MAX\_SIZE];

int res;

/\* Input two strings from user \*/

printf("Enter first string: ");

gets(str1);

printf("Enter second string: ");

gets(str2);

/\* Call the compare function to compare strings \*/

res = compare(str1, str2);

if(res == 0)

{

printf("Both strings are equal.");

}

else if(res < 0)

{

printf("First string is lexicographically smaller than second.");

}

else

{

printf("First string is lexicographically greater than second.");

}

return 0;

}

/\*\*

\* Compares two strings lexicographically.

\* Returns 0 if both strings are equal,

\* negative if first string is smaller

\* otherwise returns a positive value

\*/

int compare(char \* str1, char \* str2)

{

int i = 0;

/\* Iterate till both strings are equal \*/

while(str1[i] == str2[i])

{

if(str1[i] == '\0' && str2[i] == '\0')

break;

i++;

}

// Return the difference of current characters.

return str1[i] - str2[i];

}

#include <stdio.h>

#include <string.h>

#define MAX\_SIZE 100 // Maximum string size

int main()

{

char str1[MAX\_SIZE], str2[MAX\_SIZE];

int res;

/\* Reads two strings from user \*/

printf("Enter first string: ");

gets(str1);

printf("Enter second string: ");

gets(str2);

/\* Call strcmp() to compare both strings and stores result in res \*/

res = strcmp(str1, str2);

if(res == 0)

{

printf("Both strings are equal.");

}

else if(res == -1)

{

printf("First string is lexicographically smaller than second.");

}

else

{

printf("First string is lexicographically greater than second.");

}

return 0;

}

1. Write a C program to convert lowercase string to uppercase.

**Source Code**

#include <stdio.h>

#include <string.h>

#define MAX\_SIZE 100 // Maximum string size

int main()

{

char str[MAX\_SIZE];

/\* Input string from user \*/

printf("Enter your text : ");

gets(str);

strupr(str); // Convert to uppercase

printf("Uppercase string : %s", str);

return 0;

}

1. Write a C program to convert uppercase string to lowercase.

**Source Code**

#include <stdio.h>

#include <string.h>

#define MAX\_SIZE 100 // Maximum string size

int main()

{

char str[MAX\_SIZE];

/\* Input string from user \*/

printf("Enter any string: ");

gets(str);

strlwr(str); // Convert to lowercase

printf("Lowercase string: %s", str);

return 0;

}

1. Write a C program to toggle case of each character of a string.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum string size

/\* Toggle case function declaration \*/

void toggleCase(char \* str);

int main()

{

char str[MAX\_SIZE];

/\* Input string from user \*/

printf("Enter any string: ");

gets(str);

printf("String before toggling case: %s", str);

toggleCase(str);

printf("String after toggling case: %s", str);

return 0;

}

/\*\*

\* Toggle case of each character in given string

\*/

void toggleCase(char \* str)

{

int i = 0;

while(str[i] != '\0')

{

if(str[i]>='a' && str[i]<='z')

{

str[i] = str[i] - 32;

}

else if(str[i]>='A' && str[i]<='Z')

{

str[i] = str[i] + 32;

}

i++;

}

}

1. Write a C program to find total number of alphabets, digits or special character in a string.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum string size

int main()

{

char str[MAX\_SIZE];

int alphabets, digits, others, i;

alphabets = digits = others = i = 0;

/\* Input string from user \*/

printf("Enter any string : ");

gets(str);

/\*

\* Check each character of string for alphabet, digit or special character

\*/

while(str[i]!='\0')

{

if((str[i]>='a' && str[i]<='z') || (str[i]>='A' && str[i]<='Z'))

{

alphabets++;

}

else if(str[i]>='0' && str[i]<='9')

{

digits++;

}

else

{

others++;

}

i++;

}

printf("Alphabets = %d\n", alphabets);

printf("Digits = %d\n", digits);

printf("Special characters = %d", others);

return 0;

}

1. Write a C program to count total number of vowels and consonants in a string.

**Source Code**

#include <stdio.h>

#include <string.h>

#define MAX\_SIZE 100 // Maximum string size

int main()

{

char str[MAX\_SIZE];

int i, len, vowel, consonant;

/\* Input strings from user \*/

printf("Enter any string: ");

gets(str);

vowel = 0;

consonant = 0;

len = strlen(str);

for(i=0; i<len; i++)

{

if((str[i]>='a' && str[i]<='z') || (str[i]>='A' && str[i]<='Z'))

{

switch(str[i])

{

case 'a':

case 'e':

case 'i':

case 'o':

case 'u':

case 'A':

case 'E':

case 'I':

case 'O':

case 'U':

vowel++;

break;

default:

consonant++;

}

}

}

printf("Total number of vowel = %d\n", vowel);

printf("Total number of consonant = %d\n", consonant);

return 0;

}

1. Write a C program to count total number of words in a string.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum string size

int main()

{

char str[MAX\_SIZE];

char prevChar;

int i, words;

/\* Input string from user \*/

printf("Enter any string: ");

gets(str);

i = 0;

words = 0;

prevChar = '\0'; // The previous character of str[0] is null

/\* Runs loop infinite times \*/

while(1)

{

if(str[i]==' ' || str[i]=='\n' || str[i]=='\t' || str[i]=='\0')

{

/\*\*

\* It is a word if current character is whitespace and

\* previous character is non-white space.

\*/

if(prevChar != ' ' && prevChar != '\n' && prevChar != '\t' && prevChar != '\0')

{

words++;

}

}

/\* Make the current character as previous character \*/

prevChar = str[i];

if(str[i] == '\0')

break;

else

i++;

}

printf("Total number of words = %d", words);

return 0;

}

1. Write a C program to find reverse of a string.

**Source Code**

#include <stdio.h>

#include <string.h>

#define MAX\_SIZE 100 // Maximum string size

int main()

{

char str[MAX\_SIZE];

/\* Input string from user \*/

printf("Enter any string: ");

gets(str);

printf("Original string = %s\n", str);

/\* Find the reverse of string \*/

strrev(str);

printf("Reverse string = %s", str);

return 0;

}

1. Write a C program to check whether a string is palindrome or not.

**Source Code**

#include <stdio.h>

#include <string.h>

#define MAX\_SIZE 100 // Maximum string size

int main()

{

char str[MAX\_SIZE], reverse[MAX\_SIZE];

int flag;

/\* Input string from user \*/

printf("Enter any string: ");

gets(str);

strcpy(reverse, str); //Copies original string to reverse

strrev(reverse); //Finds the reverse of string

flag = strcmp(str, reverse); //Checks whether both are equal or not

/\* If both strings are equal \*/

if(flag == 0)

{

printf("String is Palindrome.");

}

else

{

printf("String is Not Palindrome.");

}

return 0;

}

1. Write a C program to reverse order of words in a given string.

**Source Code**

#include <stdio.h>

#include <string.h>

#define MAX\_SIZE 100 // Maximum string size

int main()

{

char str[100], reverse[100];

int len, i, index, wordStart, wordEnd;

printf("Enter any string: ");

gets(str);

len = strlen(str);

index = 0;

// Start checking of words from the end of string

wordStart = len - 1;

wordEnd = len - 1;

while(wordStart > 0)

{

// If a word is found

if(str[wordStart] == ' ')

{

// Add the word to the reverse string

i = wordStart + 1;

while(i <= wordEnd)

{

reverse[index] = str[i];

i++;

index++;

}

reverse[index++] = ' ';

wordEnd = wordStart - 1;

}

wordStart--;

}

// Finally add the last word

for(i=0; i<=wordEnd; i++)

{

reverse[index] = str[i];

index++;

}

// Add NULL character at the end of reverse string

reverse[index] = '\0';

printf("Original string \n%s\n\n", str);

printf("Reverse ordered words \n%s", reverse);

return 0;

}

1. Write a C program to find first occurrence of a character in a given string.

**Source Code**

#include <stdio.h>

#include <string.h>

#define MAX\_SIZE 100 // Maximum string size

/\* Function declaration \*/

int indexOf(const char \* str, const char toFind);

int main()

{

char str[MAX\_SIZE];

char toFind;

int index;

/\* Input string from user and character to be searched \*/

printf("Enter any string: ");

gets(str);

printf("Enter character to be searched: ");

toFind = getchar();

index = indexOf(str, toFind);

if(index == -1)

printf("'%c' not found.", toFind);

else

printf("'%c' is found at index %d.", toFind, index);

return 0;

}

/\*\*

\* Returns the first index of the given character toFind in the string.

\* If returns -1 if the given character toFind does not exists in the string.

\*/

int indexOf(const char \* str, const char toFind)

{

int i = 0;

while(str[i] != '\0')

{

if(str[i] == toFind)

return i;

i++;

}

// Return -1 as character not found

return -1;

}

1. Write a C program to find last occurrence of a character in a given string.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum string size

/\* Function declaration \*/

int lastIndexOf(const char \* str, const char toFind);

int main()

{

char str[MAX\_SIZE];

char toFind;

int index;

printf("Enter any string: ");

gets(str);

printf("Enter any character to find: ");

toFind = getchar();

index = lastIndexOf(str, toFind);

printf("\nLast index of '%c' is %d", toFind, index);

return 0;

}

/\*\*

\* Function to find last index of any character in the given string

\*/

int lastIndexOf(const char \* str, const char toFind)

{

int index = -1;

int i = 0;

while(str[i] != '\0')

{

// Update index if match is found

if(str[i] == toFind)

{

index = i;

}

i++;

}

return index;

}

1. Write a C program to search all occurrences of a character in given string.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum string size

int main()

{

char str[MAX\_SIZE];

char toSearch;

int i;

/\* Input string and character to search from user \*/

printf("Enter any string: ");

gets(str);

printf("Enter any character to search: ");

toSearch = getchar();

/\* Run loop till the last character of string \*/

i=0;

while(str[i]!='\0')

{

/\* If character is found in string \*/

if(str[i] == toSearch)

{

printf("'%c' is found at index %d\n", toSearch, i);

}

i++;

}

return 0;

}

1. Write a C program to count occurrences of a character in given string.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum string size

int main()

{

char str[MAX\_SIZE];

char toSearch;

int i, count;

/\* Input string and search character from user \*/

printf("Enter any string: ");

gets(str);

printf("Enter any character to search: ");

toSearch = getchar();

count = 0;

i=0;

while(str[i] != '\0')

{

/\*

\* If character is found in string then

\* increment count variable

\*/

if(str[i] == toSearch)

{

count++;

}

i++;

}

printf("Total occurrence of '%c' = %d", toSearch, count);

return 0;

}

1. Write a C program to find highest frequency character in a string.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum string size

#define MAX\_CHARS 255 // Maximum characters allowed

int main()

{

char str[MAX\_SIZE];

int freq[MAX\_CHARS]; // Store frequency of each character

int i = 0, max;

int ascii;

printf("Enter any string: ");

gets(str);

/\* Initializes frequency of all characters to 0 \*/

for(i=0; i<MAX\_CHARS; i++)

{

freq[i] = 0;

}

/\* Finds frequency of each characters \*/

i=0;

while(str[i] != '\0')

{

ascii = (int)str[i];

freq[ascii] += 1;

i++;

}

/\* Finds maximum frequency \*/

max = 0;

for(i=0; i<MAX\_CHARS; i++)

{

if(freq[i] > freq[max])

max = i;

}

printf("Maximum occurring character is '%c' = %d times.", max, freq[max]);

return 0;

}

1. Write a C program to find lowest frequency character in a string.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum string size

#define MAX\_CHARS 255 // Maximum characters allowed

int main()

{

char str[MAX\_SIZE];

int freq[MAX\_CHARS]; //Stores frequency of each character

int i = 0, min;

int ascii;

printf("Enter any string: ");

gets(str);

/\* Initialize frequency of all characters to 0 \*/

for(i=0; i<MAX\_CHARS; i++)

{

freq[i] = 0;

}

/\* Finds frequency of each characters \*/

i=0;

while(str[i] != '\0')

{

ascii = (int)str[i];

freq[ascii] += 1;

i++;

}

/\* Finds minimum frequency \*/

min = 0;

for(i=0; i<MAX\_CHARS; i++)

{

if(freq[i] != 0)

{

if(freq[min] == 0 || freq[i] < freq[min])

min = i;

}

}

printf("Minimum occurring character is '%c' = %d.", min, freq[min]);

return 0;

}

1. Write a C program to count frequency of each character in a string.

**Source Code**

#include <stdio.h>

#include <string.h>

#define MAX\_SIZE 100 // Maximum string size

int main()

{

char str[MAX\_SIZE];

int i, len;

int freq[26];

/\* Input string from user \*/

printf("Enter any string: ");

gets(str);

len = strlen(str);

/\* Initialize frequency of each character to 0 \*/

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

{

freq[i] = 0;

}

/\* Find total number of occurrences of each character \*/

for(i=0; i<len; i++)

{

/\* If the current character is lowercase alphabet \*/

if(str[i]>='a' && str[i]<='z')

{

freq[str[i] - 97]++;

}

else if(str[i]>='A' && str[i]<='Z')

{

freq[str[i] - 65]++;

}

}

/\* Print the frequency of all characters in the string \*/

printf("\nFrequency of all characters in the given string: \n");

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

{

/\* If current character exists in given string \*/

if(freq[i] != 0)

{

printf("'%c' = %d\n", (i + 97), freq[i]);

}

}

return 0;

}

1. Write a C program to remove first occurrence of a character from string.

**Source Code**

#include <stdio.h>

#include <string.h>

#define MAX\_SIZE 100 // Maximum string size

/\* Function declaration \*/

void removeFirst(char \*, const char);

int main()

{

char str[MAX\_SIZE];

char toRemove;

printf("Enter any string: ");

gets(str);

printf("Enter character to remove from string: ");

toRemove = getchar();

removeFirst(str, toRemove);

printf("String after removing first '%c' : %s", toRemove, str);

return 0;

}

/\*\*

\* Function to remove first occurrence of a character from the string.

\*/

void removeFirst(char \* str, const char toRemove)

{

int i = 0;

int len = strlen(str);

/\* Run loop till the first occurrence of the character is not found \*/

while(i<len && str[i]!=toRemove)

i++;

/\* Shift all characters right to the position found above, to one place left \*/

while(i < len)

{

str[i] = str[i+1];

i++;

}

}

1. Write a C program to remove last occurrence of a character from string.

**Source Code**

#include <stdio.h>

#include <string.h>

#define MAX\_SIZE 100 // Maximum string size

/\* Function declaration \*/

void removeLast(char \*, const char);

int main()

{

char str[MAX\_SIZE];

char toRemove;

printf("Enter any string: ");

gets(str);

printf("Enter character to remove from string: ");

toRemove = getchar();

removeLast(str, toRemove);

printf("String after removing last '%c' : %s", toRemove, str);

return 0;

}

/\*\*

\* Function to remove last occurrence of a character from the string.

\*/

void removeLast(char \* str, const char toRemove)

{

int i, lastPosition;

int len = strlen(str);

/\* Assume that character does not exist in string \*/

lastPosition = -1;

i=0;

while(i<len)

{

if(str[i] == toRemove)

{

lastPosition = i;

}

i++;

}

/\* If character exists in string \*/

if(lastPosition != -1)

{

i = lastPosition;

/\*

\* Shift all characters right to the position found above to left

\*/

while(i<len)

{

str[i] = str[i+1];

i++;

}

}

}

1. Write a C program to remove all occurrences of a character from string.

**Source Code**

#include <stdio.h>

#include <string.h>

#define MAX\_SIZE 100 // Maximum string size

/\*\* Function declaration \*/

void removeAll(char \*, const char);

int main()

{

char str[MAX\_SIZE];

char toRemove;

printf("Enter any string: ");

gets(str);

printf("Enter character to remove from string: ");

toRemove = getchar();

removeAll(str, toRemove);

printf("String after removing '%c': %s", toRemove, str);

return 0;

}

/\*\*

\* Function to remove all occurrences of a character from the string.

\*/

void removeAll(char \* str, const char toRemove)

{

int i, j;

int len = strlen(str);

for(i=0; i<len; i++)

{

/\*

\* If the character to remove is found then shift all characters to one

\* place left and decrement the length of string by 1.

\*/

if(str[i] == toRemove)

{

for(j=i; j<len; j++)

{

str[j] = str[j+1];

}

len--;

// If a character is removed then make sure i doesn't increments

i--;

}

}

}

1. Write a C program to remove all repeated characters from a given string.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum string size

/\* Function declarations \*/

void removeDuplicates(char \* str);

void removeAll(char \* str, const char toRemove, int index);

int main()

{

char str[MAX\_SIZE];

/\* Input string from user \*/

printf("Enter any string: ");

gets(str);

printf("String before removing duplicates: %s\n", str);

removeDuplicates(str);

printf("String after removing duplicates: %s\n", str);

return 0;

}

/\*\*

\* Remove all duplicate characters from the given string

\*/

void removeDuplicates(char \* str)

{

int i = 0;

while(str[i] != '\0')

{

/\* Remove all duplicate of character string[i] \*/

removeAll(str, str[i], i + 1);

i++;

}

}

/\*\*

\* Remove all occurrences of a given character from string.

\*/

void removeAll(char \* str, const char toRemove, int index)

{

int i;

while(str[index] != '\0')

{

/\* If duplicate character is found \*/

if(str[index] == toRemove)

{

/\* Shift all characters from current position to one place left \*/

i = index;

while(str[i] != '\0')

{

str[i] = str[i + 1];

i++;

}

}

else

{

index++;

}

}

}

1. Write a C program to replace first occurrence of a character with another in a string.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum string size

/\* Function declaration \*/

void replaceFirst(char \* str, char oldChar, char newChar);

int main()

{

char str[MAX\_SIZE], oldChar, newChar;

printf("Enter any string: ");

gets(str);

printf("Enter character to replace: ");

oldChar = getchar();

// Used to skip extra ENTER character

getchar();

printf("Enter character to replace '%c' with: ", oldChar);

newChar = getchar();

printf("\nString before replacing: %s\n", str);

replaceFirst(str, oldChar, newChar);

printf("String after replacing first '%c' with '%c' : %s", oldChar, newChar, str);

return 0;

}

/\*\*

\* Replace first occurrence of a character with

\* another in given string.

\*/

void replaceFirst(char \* str, char oldChar, char newChar)

{

int i=0;

/\* Run till end of string \*/

while(str[i] != '\0')

{

/\* If an occurrence of character is found \*/

if(str[i] == oldChar)

{

str[i] = newChar;

break;

}

i++;

}

}

1. Write a C program to replace last occurrence of a character with another in a string.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum string size

/\* Function declaration \*/

void replaceLast(char \* str, char oldChar, char newChar);

int main()

{

char str[MAX\_SIZE], oldChar, newChar;

printf("Enter any string: ");

gets(str);

printf("Enter character to replace: ");

oldChar = getchar();

// Dummy getchar() to eliminate extra ENTER character

getchar();

printf("Enter character to replace '%c' with: ", oldChar);

newChar = getchar();

printf("\nString before replacing: \n%s", str);

replaceLast(str, oldChar, newChar);

printf("\n\nString after replacing '%c' with '%c': \n%s", oldChar, newChar, str);

return 0;

}

/\*\*

\* Replace last occurrence of a character with

\* another in given string.

\*/

void replaceLast(char \* str, char oldChar, char newChar)

{

int i, lastIndex;

lastIndex = -1;

i = 0;

/\* Run till end of string \*/

while(str[i] != '\0')

{

/\* If an occurrence of character is found \*/

if(str[i] == oldChar)

{

lastIndex = i;

}

i++;

}

if(lastIndex != -1)

{

str[lastIndex] = newChar;

}

}

1. Write a C program to replace all occurrences of a character with another in a string.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum string size

/\* Function declaration \*/

void replaceAll(char \* str, char oldChar, char newChar);

int main()

{

char str[MAX\_SIZE], oldChar, newChar;

printf("Enter any string: ");

gets(str);

printf("Enter character to replace: ");

oldChar = getchar();

// Dummy getchar() to eliminate extra ENTER character

getchar();

printf("Enter character to replace '%c' with: ", oldChar);

newChar = getchar();

printf("\nString before replacing: \n%s", str);

replaceAll(str, oldChar, newChar);

printf("\n\nString after replacing '%c' with '%c' : \n%s", oldChar, newChar, str);

return 0;

}

/\*\*

\* Replace all occurrence of a character in given string.

\*/

void replaceAll(char \* str, char oldChar, char newChar)

{

int i = 0;

/\* Run till end of string \*/

while(str[i] != '\0')

{

/\* If occurrence of character is found \*/

if(str[i] == oldChar)

{

str[i] = newChar;

}

i++;

}

}

1. Write a C program to find first occurrence of a word in a given string.

**Source Code**

#include <stdio.h>

#include <string.h>

#define MAX\_SIZE 100 // Maximum string size

int main()

{

char str[MAX\_SIZE], word[MAX\_SIZE];

int i, index, found = 0;

/\* Input string and word from user \*/

printf("Enter any string: ");

gets(str);

printf("Enter word to be searched: ");

gets(word);

/\* Run loop from start to end of string \*/

index = 0;

while(str[index] != '\0')

{

/\* If first character of word matches with the given string \*/

if(str[index] == word[0])

{

/\* Match entire word with current found index \*/

i=0;

found = 1;

while(word[i] != '\0')

{

if(str[index + i] != word[i])

{

found = 0;

break;

}

i++;

}

}

/\* If the word is found then get out of loop \*/

if(found == 1)

{

break;

}

index++;

}

/\* Print success message if the word is found \*/

if(found == 1)

{

printf("\n'%s' is found at index %d.", word, index);

}

else

{

printf("\n'%s' is not found.", word);

}

return 0;

}

1. Write a C program to find last occurrence of a word in a given string.

**Source Code**

#include <stdio.h>

#include <string.h>

#define MAX\_SIZE 100 // Maximum string size

int main()

{

char str[MAX\_SIZE];

char word[MAX\_SIZE];

int i, j, index, found;

int strLen, wordLen;

/\* Input string and word from user \*/

printf("Enter any string: ");

gets(str);

printf("Enter any word to search: ");

gets(word);

index = -1;

strLen = strlen(str); // Find length of string

wordLen = strlen(word); // Find length of word

/\*

\* Runs a loop from starting index of string to

\* length of string - word length

\*/

for(i=0; i<=strLen - wordLen; i++)

{

// Match word at current position

found = 1;

for(j=0; j<wordLen; j++)

{

//If word is not matched

if(str[i + j] != word[j])

{

found = 0;

break;

}

}

// If word have been found then store the current found index

if(found == 1)

{

index = i;

}

}

if(index == -1)

{

printf("\n'%s' not found.", word);

}

else

{

printf("\nLast index of '%s' = %d", word, index);

}

return 0;

}

1. Write a C program to search all occurrences of a word in given string.

**Source Code**

#include <stdio.h>

#include <string.h>

#define MAX\_SIZE 100 // Maximum string size

int main()

{

char str[MAX\_SIZE];

char word[MAX\_SIZE];

int i, j, found;

int strLen, wordLen;

/\* Input string and word from user \*/

printf("Enter any string: ");

gets(str);

printf("Enter any word to search: ");

gets(word);

strLen = strlen(str); // Find length of string

wordLen = strlen(word); // Find length of word

/\*

\* Run a loop from starting index of string to

\* length of string - word length

\*/

for(i=0; i<strLen - wordLen; i++)

{

// Match word at current position

found = 1;

for(j=0; j<wordLen; j++)

{

// If word is not matched

if(str[i + j] != word[j])

{

found = 0;

break;

}

}

// If word have been found then print found message

if(found == 1)

{

printf("'%s' found at index: %d \n", word, i);

}

}

return 0;

}

1. Write a C program to count occurrences of a word in a given string.

**Source Code**

#include <stdio.h>

#include <string.h>

#define MAX\_SIZE 100 // Maximum string size

/\* Function declaration \*/

int countOccurrences(char \* str, char \* toSearch);

int main()

{

char str[MAX\_SIZE];

char toSearch[MAX\_SIZE];

int count;

/\* Input string and word from user \*/

printf("Enter any string: ");

gets(str);

printf("Enter word to search occurrences: ");

gets(toSearch);

count = countOccurrences(str, toSearch);

printf("Total occurrences of '%s': %d", toSearch, count);

return 0;

}

/\*\*

\* Get, total number of occurrences of a word in a string

\*/

int countOccurrences(char \* str, char \* toSearch)

{

int i, j, found, count;

int stringLen, searchLen;

stringLen = strlen(str); // length of string

searchLen = strlen(toSearch); // length of word to be searched

count = 0;

for(i=0; i <= stringLen-searchLen; i++)

{

/\* Match word with string \*/

found = 1;

for(j=0; j<searchLen; j++)

{

if(str[i + j] != toSearch[j])

{

found = 0;

break;

}

}

if(found == 1)

{

count++;

}

}

return count;

}

1. Write a C program to remove first occurrence of a word from string.

**Source Code**

#include <stdio.h>

#include <string.h>

#define MAX\_SIZE 100 // Maximum string size

/\*\* Function declaration \*/

void removeFirst(char \* str, const char \* toRemove);

int main()

{

char str[MAX\_SIZE];

char toRemove[MAX\_SIZE];

/\* Input string and word to be removed from user \*/

printf("Enter any string: ");

gets(str);

printf("Enter string to be removed: ");

gets(toRemove);

removeFirst(str, toRemove);

printf("\nString after removing '%s': \n%s", toRemove, str);

return 0;

}

/\*\*

\* Remove first occurrence of a word from string

\*/

void removeFirst(char \* str, const char \* toRemove)

{

int i, j;

int len, removeLen;

int found = 0;

len = strlen(str);

removeLen = strlen(toRemove);

for(i=0; i<len; i++)

{

found = 1;

for(j=0; j<removeLen; j++)

{

if(str[i+j] != toRemove[j])

{

found = 0;

break;

}

}

/\* If word has been found then remove it by shifting characters \*/

if(found == 1)

{

for(j=i; j<=len-removeLen; j++)

{

str[j] = str[j + removeLen];

}

// Terminate from loop so only first occurrence is removed

break;

}

}

}

1. Write a C program to remove last occurrence of a word in given string.

**Source Code**

#include <stdio.h>

#include <string.h>

#define MAX\_SIZE 100 // Maximum string size

int main()

{

char str[MAX\_SIZE];

char word[MAX\_SIZE];

int i, j, found, index;

int stringLen, wordLen;

/\* Input string and word from user \*/

printf("Enter any string: ");

gets(str);

printf("Enter word to remove: ");

gets(word);

stringLen = strlen(str); // Length of string

wordLen = strlen(word); // Length of word

/\*

\* Run loop from start to end of string - word length

\*/

index = -1;

for(i=0; i<stringLen - wordLen; i++)

{

// Match word at current position

found = 1;

for(j=0; j<wordLen; j++)

{

// If word is not matched

if(str[i+j] != word[j])

{

found = 0;

break;

}

}

// If word is found then update index

if(found == 1)

{

index = i;

}

}

// If word not found

if(index == -1)

{

printf("'%s' not found.");

}

else

{

/\*

\* Shift all characters from right to left

\*/

for(i=index; i <= stringLen - wordLen; i++)

{

str[i] = str[i + wordLen];

}

printf("String after removing last '%s': \n%s", word, str);

}

return 0;

}

1. Write a C program to remove all occurrence of a word in given string.

**Source Code**

#include <stdio.h>

#include <string.h>

#define MAX\_SIZE 100 // Maximum string size

/\* Function declaration \*/

void removeAll(char \* str, char \* toRemove);

int main()

{

char str[MAX\_SIZE];

char toRemove[MAX\_SIZE];

/\* Input string and word from user \*/

printf("Enter any string: ");

gets(str);

printf("Enter word to remove: ");

gets(toRemove);

printf("String before removing '%s' : \n%s", toRemove, str);

removeAll(str, toRemove);

printf("\n\nString after removing '%s' : \n%s", toRemove, str);

return 0;

}

/\*\*

\* Remove all occurrences of a given word in string.

\*/

void removeAll(char \* str, char \* toRemove)

{

int i, j, stringLen, toRemoveLen;

int found;

stringLen = strlen(str); // Length of string

toRemoveLen = strlen(toRemove); // Length of word to remove

for(i=0; i <= stringLen - toRemoveLen; i++)

{

/\* Match word with string \*/

found = 1;

for(j=0; j<toRemoveLen; j++)

{

if(str[i + j] != toRemove[j])

{

found = 0;

break;

}

}

/\* If it is not a word \*/

if(str[i + j] != ' ' && str[i + j] != '\t' && str[i + j] != '\n' && str[i + j] != '\0')

{

found = 0;

}

/\*

\* If word is found then shift all characters to left

\* and decrement the string length

\*/

if(found == 1)

{

for(j=i; j<=stringLen - toRemoveLen; j++)

{

str[j] = str[j + toRemoveLen];

}

stringLen = stringLen - toRemoveLen;

// We will match next occurrence of word from current index.

i--;

}

}

}

1. Write a C program to trim leading white space characters from given string.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum string size

/\* Function declaration \*/

void trimLeading(char \* str);

int main()

{

char str[MAX\_SIZE];

/\* Input string from user \*/

printf("Enter any string: ");

gets(str);

printf("\nString before trimming leading whitespace: \n%s", str);

trimLeading(str);

printf("\n\nString after trimming leading whitespace: \n%s", str);

return 0;

}

/\*\*

\* Remove leading whitespace characters from string

\*/

void trimLeading(char \* str)

{

int index, i, j;

index = 0;

/\* Find last index of whitespace character \*/

while(str[index] == ' ' || str[index] == '\t' || str[index] == '\n')

{

index++;

}

if(index != 0)

{

/\* Shit all trailing characters to its left \*/

i = 0;

while(str[i + index] != '\0')

{

str[i] = str[i + index];

i++;

}

str[i] = '\0'; // Make sure that string is NULL terminated

}

}

1. Write a C program to trim trailing white space characters from given string.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum string size

/\* Function declaration \*/

void trimTrailing(char \* str);

int main()

{

char str[MAX\_SIZE];

/\* Input string from user \*/

printf("Enter any string: ");

gets(str);

printf("\nString before trimming trailing white space: \n'%s'", str);

trimTrailing(str);

printf("\n\nString after trimming trailing white spaces: \n'%s'", str);

return 0;

}

/\*\*

\* Remove trailing white space characters from string

\*/

void trimTrailing(char \* str)

{

int index, i;

/\* Set default index \*/

index = -1;

/\* Find last index of non-white space character \*/

i = 0;

while(str[i] != '\0')

{

if(str[i] != ' ' && str[i] != '\t' && str[i] != '\n')

{

index= i;

}

i++;

}

/\* Mark next character to last non-white space character as NULL \*/

str[index + 1] = '\0';

}

1. Write a C program to trim both leading and trailing white space characters from given string.

**Source Code**

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum string size

/\* Function declaration \*/

void trim(char \* str);

int main()

{

char str[MAX\_SIZE];

/\* Input string from user \*/

printf("Enter any string: ");

gets(str);

printf("\nString before trimming white space: \n'%s'", str);

trim(str);

printf("\n\nString after trimming white space: \n'%s'", str);

return 0;

}

/\*\*

\* Remove leading and trailing white space characters

\*/

void trim(char \* str)

{

int index, i;

/\*

\* Trim leading white spaces

\*/

index = 0;

while(str[index] == ' ' || str[index] == '\t' || str[index] == '\n')

{

index++;

}

/\* Shift all trailing characters to its left \*/

i = 0;

while(str[i + index] != '\0')

{

str[i] = str[i + index];

i++;

}

str[i] = '\0'; // Terminate string with NULL

/\*

\* Trim trailing white spaces

\*/

i = 0;

index = -1;

while(str[i] != '\0')

{

if(str[i] != ' ' && str[i] != '\t' && str[i] != '\n')

{

index = i;

}

i++;

}

/\* Mark the next character to last non white space character as NULL \*/

str[index + 1] = '\0';

}

1. Write a C program to remove all extra blank spaces from given string.

**Source Code**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 100 // Maximum string size

/\* Function declaration \*/

char \* removeBlanks(const char \* str);

int main()

{

char str[MAX\_SIZE];

char \* newString;

printf("Enter any string: ");

gets(str);

printf("\nString before removing blanks: \n'%s'", str);

newString = removeBlanks(str);

printf("\n\nString after removing blanks: \n'%s'", newString);

return 0;

}

/\*\*

\* Removes extra blank spaces from the given string

\* and returns a new string with single blank spaces

\*/

char \* removeBlanks(const char \* str)

{

int i, j;

char \* newString;

newString = (char \*)malloc(MAX\_SIZE);

i = 0;

j = 0;

while(str[i] != '\0')

{

/\* If blank space is found \*/

if(str[i] == ' ')

{

newString[j] = ' ';

j++;

/\* Skip all consecutive spaces \*/

while(str[i] == ' ')

i++;

}

newString[j] = str[i];

i++;

j++;

}

// NULL terminate the new string

newString[j] = '\0';

return newString;

}