**CSA06- DAA- LAB PROGRAMS**

1. Write a c-program to print Fibonacci series using recursion.

**Code:**

**Output:**

#include<stdio.h>

int fib(int n){

if(n==0){

return 0;

}

else if(n==1){

return 1;

}

else{

return (fib(n-1)+fib(n-2));

}

}

int main(){

int num;

printf("enter num:");

scanf("%d",&num);

printf("fibonacci series: \n ");

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

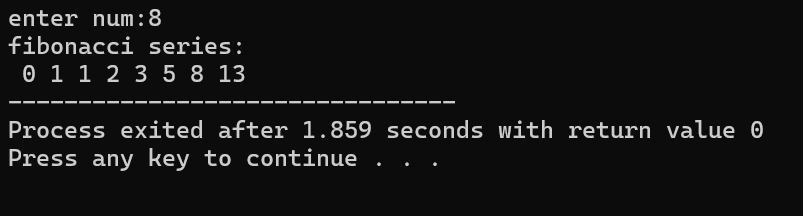
printf("%d ",fib(i));

}

return 0;

}

**Output:**



1. Write a c-program to check the given number is Armstrong or not using recursion.

**Code:**

#include <stdio.h>

#include <math.h>

int countDigits(int n) {

if (n == 0)

return 0;

return 1 + countDigits(n / 10);

}

int isArmstrongRec(int n, int numDigits) {

if (n == 0)

return 0;

int digit = n % 10;

return pow(digit, numDigits) + isArmstrongRec(n / 10, numDigits);

}

int isArmstrong(int n) {

int numDigits = countDigits(n);

return (n == isArmstrongRec(n, numDigits));

}

int main() {

int num;

printf("Enter a number: ");

scanf("%d", &num);

printf("digits:",countDigits(num));

if (isArmstrong(num))

printf("%d is an Armstrong number.\n", num);

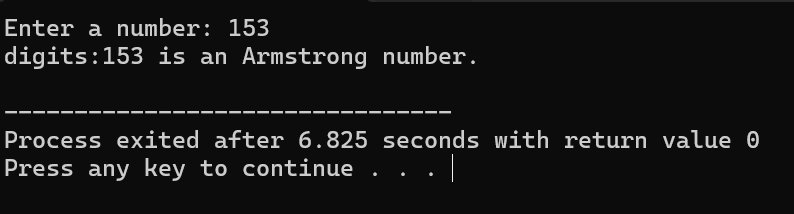
else

printf("%d is not an Armstrong number.\n", num);

return 0;

}

**Output:**



1. Write a c-program to find GCD of two numbers.

**Code:**

#include <stdio.h>

int gcd(int a, int b) {

if (b == 0)

return a;

return gcd(b, a % b);

}

int main() {

int num1, num2;

printf("Enter two numbers: ");

scanf("%d %d", &num1, &num2);

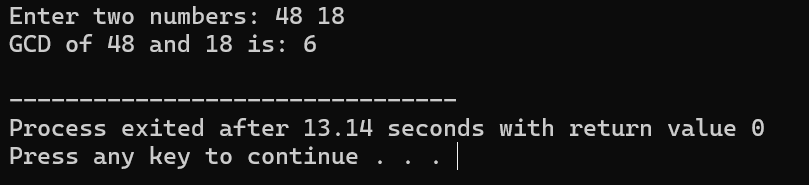
int result = gcd(num1, num2);

printf("GCD of %d and %d is: %d\n", num1, num2, result);

return 0;

}

**Output:**



1. Write a c-program to find largest element in the array.

**Code:**

#include<stdio.h>

int main(){

int n;

printf("enter size: ");

scanf("%d",&n);

int arr[n];

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

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

}

int max=arr[0];

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

if(arr[i]>max){

max=arr[i];

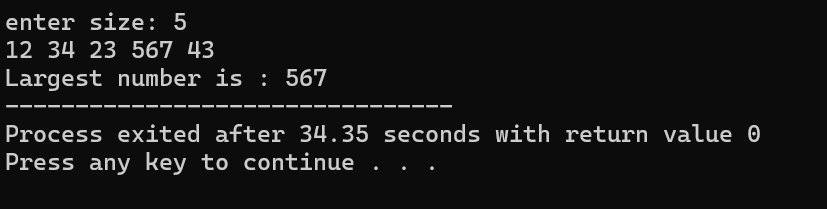
}

}

printf("Largest number is : %d",max);

}

**Output:**



1. Write a c-program to find factorial of given number.

**Code:**

#include <stdio.h>

int fact(int n){

if(n==0||n==1){

return 1;

}

return n\*fact(n-1);

}

int main(){

printf("enter Number: ");

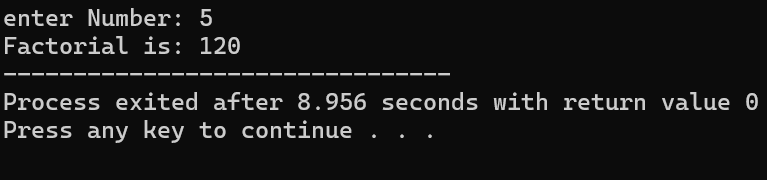
int num;

scanf("%d",&num);

printf("Factorial is: %d",fact(num));

}

**Output:**



1. Write a c-program to check the given number is prime or not.

**Code:**

#include <stdio.h>

int isprime(int a,int i){

if (a<=2){

return(a==2)? 1:0;

}

if(a%i==0){

return 0;

}

if(i\*i>a){

return 1;

}

return (isprime(a,i+1));

}

int main(){

printf("Enter a number: ");

int num;

scanf("%d",&num);

if(isprime(num,2)){

printf("Prime number!");

}

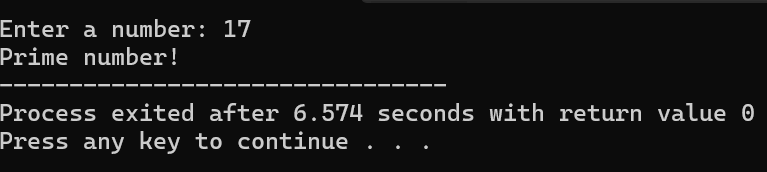
else{

printf("Not prime number");

}

}

**Output:**



1. Write a c-program to sort the list using selection sort.

**Code:**

#include <stdio.h>

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

int i, j, minIndex, temp;

for (i = 0; i < n-1; 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;

}

}

}

int main() {

int n, i;

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

scanf("%d", &n);

int arr[n];

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

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

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

}

selectionSort(arr, n);

printf("Sorted array: ");

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

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

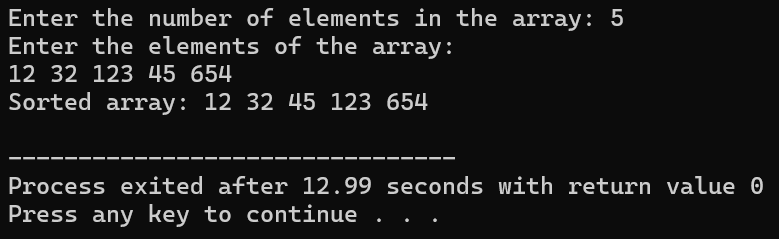
}

printf("\n");

return 0;

}

**Output:**

****

1. Write a c-program to sort the array using bubble sort.

**Code:**

#include <stdio.h>

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

int i, j, temp;

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

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

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

temp = arr[j];

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

arr[j+1] = temp;

}

}

}

}

int main() {

int n, i;

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

scanf("%d", &n);

int arr[n];

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

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

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

}

bubbleSort(arr, n);

printf("Sorted array: ");

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

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

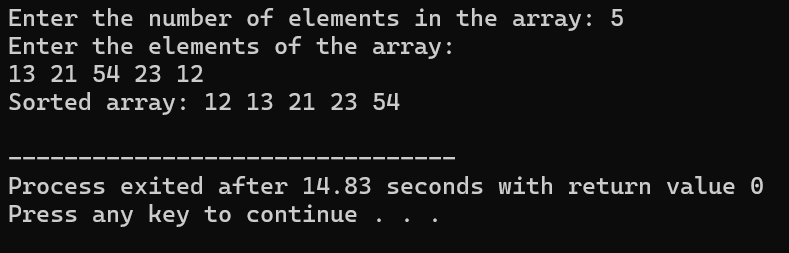
}

printf("\n");

return 0;

}

**Output:**

****

1. Write a c-program to find time complexity for matrix multiplication.

**Code:**

#include <stdio.h>

void multiplyMatrices(int mat1[][10], int mat2[][10], int res[][10], int r1, int c1, int c2) {

int i, j, k;

for (i = 0; i < r1; i++) {

for (j = 0; j < c2; j++) {

res[i][j] = 0;

}

}

int operationCount = 0;

for (i = 0; i < r1; i++){

for (j = 0; j < c2; j++) {

for (k = 0; k < c1; k++) {

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

operationCount++;

}

}

}

printf("Number of operations performed (Time Complexity): %d\n", operationCount);

}

int main() {

int r1, c1, r2, c2;

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

scanf("%d %d", &r1, &c1);

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

scanf("%d %d", &r2, &c2);

if (c1 != r2) {

printf("Matrix multiplication not possible.\n");

return 0;

}

int mat1[10][10], mat2[10][10], res[10][10];

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

for (int i = 0; i < r1; i++) {

for (int j = 0; j < c1; j++) {

scanf("%d", &mat1[i][j]);

}

}

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

for (int i = 0; i < r2; i++) {

for (int j = 0; j < c2; j++) {

scanf("%d", &mat2[i][j]);

}

}

multiplyMatrices(mat1, mat2, res, r1, c1, c2);

printf("Resultant Matrix:\n");

for (int i = 0; i < r1; i++) {

for (int j = 0; j < c2; j++) {

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

}

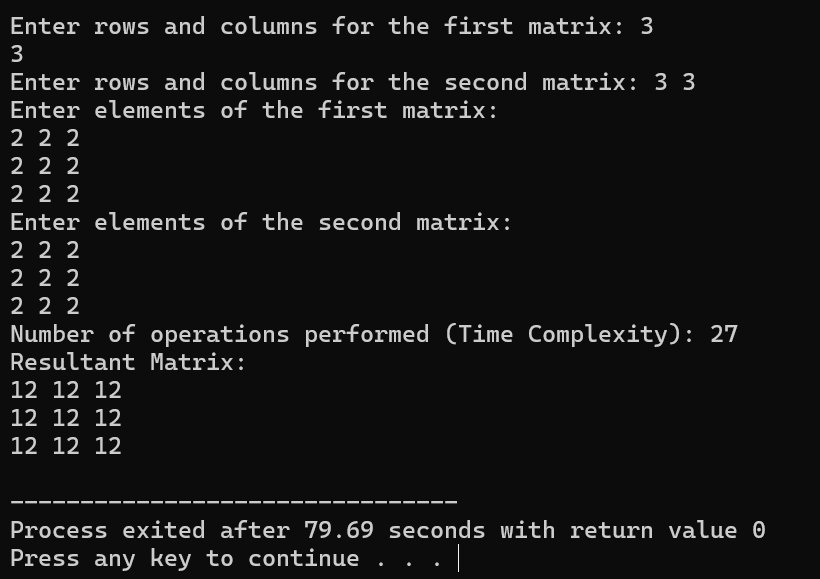
printf("\n");

}

return 0;

}

**Output:**

****

1. Write a c-program to check given number is palindrome or not.

**Code:**

#include <stdio.h>

#include <string.h>

int palindrome(char s[],int st,int en){

if(st>=en){

return 1;

}

if(s[st]!=s[en]){

return 0;

}

return (palindrome(s,st+1,en-1));

}

int main(){

char s[20];

printf("enter a string: ");

scanf("%s",s);

int n;

n=strlen(s);

int end=n-1;

if(palindrome(s,0,end)){

printf("Palindrome!!");

}

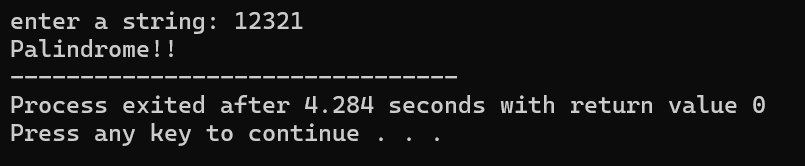
else{

printf("Not palindrome");

}

}

**Output:**

****

1. Write a c-program to copy one string to another.

**Code:**

#include<stdio.h>

void copystring(const char \*source ,char \*destination){

int i;

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

destination[i]=source[i];

}

destination[i]='\0';

}

int main(){

char source[]="hello world";

char destination[1000];

copystring(source,destination);

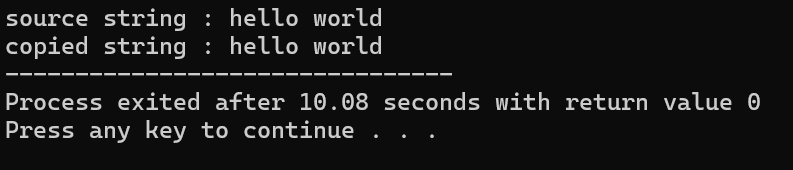
printf("source string : %s\n",source);

printf("copied string : %s",destination);

return 0;

}

**Output:**

****

1. Write a c-program to perform binary search.

**Code:**

#include<stdio.h>

void sorted(int array[], int n){

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

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

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

int temp=array[j];

array[j]=array[j+1];

array[j+1]=temp;

}

}

}

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

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

}

printf("]");

}

int binarysearch(int array[], int n,int target){

int low=0,high=n-1;

while(low<=high){

int mid=(low+high)/2;

if(array[mid]==target){

return mid;

}

else if(array[mid]>target){

high=mid-1;

}

else{

low=mid+1;

}

}

return -1;

}

int main(){

int n,target;

printf("enter no of elements in array:");

scanf("%d",&n);

int array[n],i,j;

printf("enter elements in array:");

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

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

}

printf("input array=[");

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

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

}

printf("]");

printf("\nenter target element:");

scanf("%d",&target);

printf("sorted array:[");

sorted(array,n);

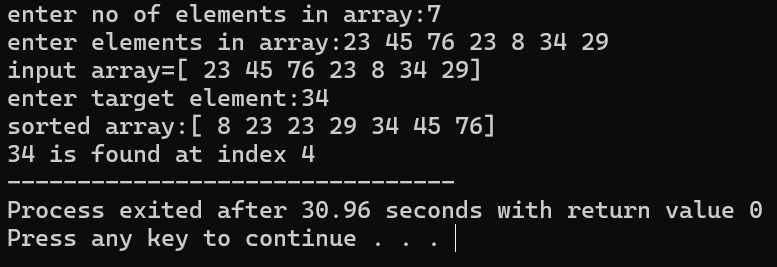
int result=binarysearch(array,n,target);

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

return 0;

}

**Output:**

****

1. Write a c-program to print reverse of a string.

**Code:**

#include<stdio.h>

#include<string.h>

void reversestring(char s[]){

int len=strlen(s);

int start=0,end=len-1;

while(start<end){

int temp=s[start];

s[start]=s[end];

s[end]=temp;

start++;

end--;

}

}

int main(){

char s[100];

printf("enter a string:");

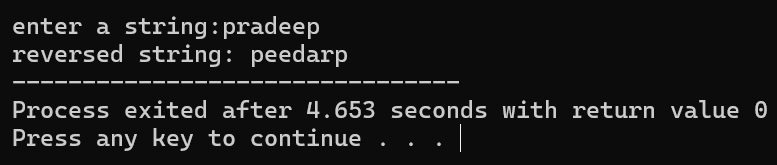
scanf("%s",&s);

reversestring(s);

printf("reversed string: %s",s);

}

**Output:**

****

1. Write a c-program to print Fibonacci series using recursion.

**Code:**

#include<stdio.h>

void stringlen(char s[]){

int i,count=0;

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

if(s[i]!='\n'){

count++;

}

}

printf("length of string %s is %d",s,count);

}

int main(){

char str[100];

printf("enter the string:");

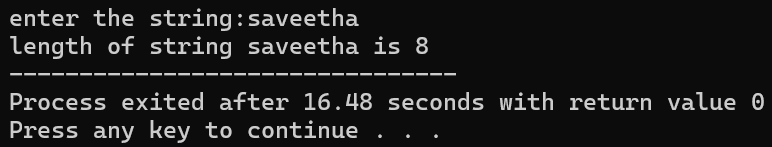
scanf("%s",&str);

stringlen(str);

return 0;

}

**Output:**

****

1. Write a c-program to perform Strassen matrix multiplication.

**Code:**

#include <stdio.h>

#include <stdlib.h>

#define MAX 4

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];

}

}

}

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];

}

}

}

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];

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];

}

}

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);

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);

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];

}

}

}

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++) {

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

}

}

}

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;

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

inputMatrix(A, size);

inputMatrix(B, size);

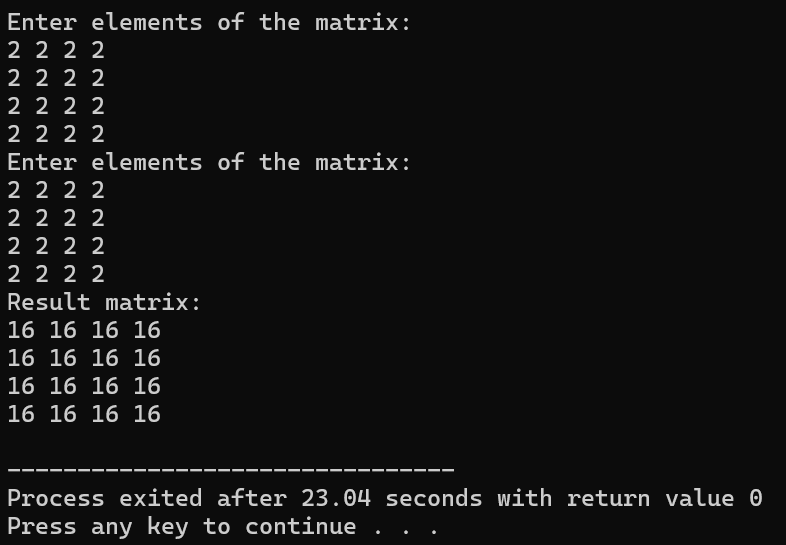
strassen(A, B, C, size);

displayMatrix(C, size);

return 0;

}

**Output:**



1. Write a c-program to perform merge sort.

**Code:**

#include <stdio.h>

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

int n1 = mid - left + 1;

int n2 = right - mid;

int L[n1], R[n2];

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];

}

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++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

void mergeSort(int arr[], int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

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

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

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

}

printf("\n");

}

int main() {

int arrSize;

printf("Enter the number of elements: ");

scanf("%d", &arrSize);

int arr[arrSize];

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);

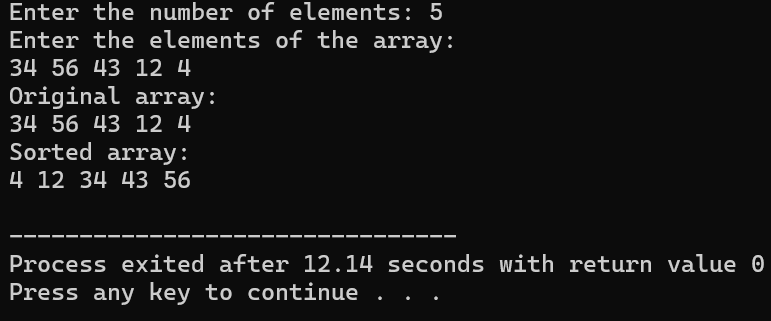
printf("Sorted array: \n");

printArray(arr, arrSize);

return 0;

}

**Output:**

****

1. Write a c-program to find max and min in the given list using divide and conquer.

**Code:**

#include <stdio.h>

void findMaxMin(int arr[], int low, int high, int \*max, int \*min) {

if (low == high) {

\*max = arr[low];

\*min = arr[low];

}

else if (high == low + 1) {

if (arr[low] > arr[high]) {

\*max = arr[low];

\*min = arr[high];

} else {

\*max = arr[high];

\*min = arr[low];

}

}

else {

int mid = (low + high) / 2;

int max1, min1, max2, min2;

findMaxMin(arr, low, mid, &max1, &min1);

findMaxMin(arr, mid + 1, high, &max2, &min2);

\*max = (max1 > max2) ? max1 : max2;

\*min = (min1 < min2) ? min1 : min2;

}

}

int main() {

int n,arr[n],max, min;

printf("enter the number of elements in array:");

scanf("%d",&n);

printf("enter the elements of the array:\n");

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

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

}

findMaxMin(arr, 0, n - 1, &max, &min);

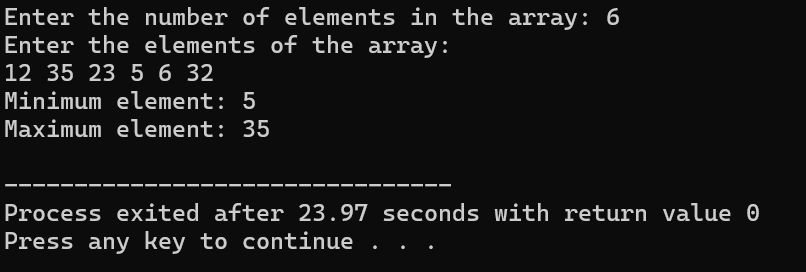
printf("Maximum value in the list: %d\n", max);

printf("Minimum value in the list: %d\n", min);

return 0;

}

**Output:**

****

1. Write a c-program to print prime numbers from 1-100.

**Code:**

#include <stdio.h>

#include <stdbool.h>

bool isPrime(int num) {

if (num <= 1) {

return false;

}

for (int i = 2; i \* i <= num; i++) {

if (num % i == 0) {

return false;

}

}

return true;

}

int main() {

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

for (int i = 1; i <= 100; i++) {

if (isPrime(i)) {

printf("%d ", i);

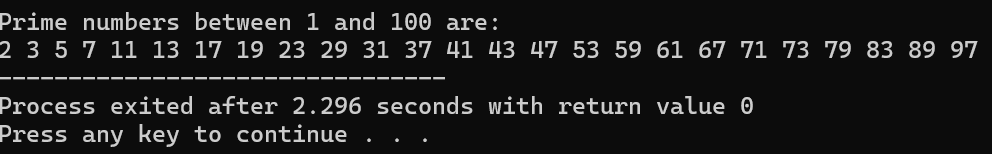
}

}

return 0;

}

**Output:**

****

1. Write a c-program to perform knapsack problem using greedy techniques.

**Code:**

#include <stdio.h>

#include <stdlib.h>

struct Item {

int value;

int weight;

float 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);

}

float knapsack(struct Item items[], int n, int capacity) {

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) {

remainingCapacity -= items[i].weight;

totalValue += items[i].value;

} else {

totalValue += items[i].value \* ((float)remainingCapacity / items[i].weight);

break;

}

}

return totalValue;

}

int main() {

int n, capacity;

printf("Enter the number of items: ");

scanf("%d", &n);

printf("Enter the capacity of the knapsack: ");

scanf("%d", &capacity);

struct Item items[n];

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;

}

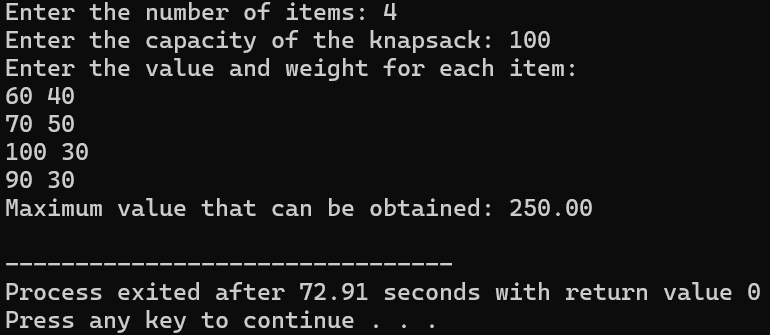
float maxValue = knapsack(items, n, capacity);

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

return 0;

}

**Output:**

****

1. Write a c-program to perform MST using greedy techniques(prims).

**Code:**

#include <stdio.h>

#include <limits.h>

#include <stdbool.h>

#define V 5

int minKey(int key[], bool mstSet[]) {

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++)

if (mstSet[v] == false && key[v] < min) {

min = key[v];

min\_index = v;

}

return min\_index;

}

void printMST(int parent[], int graph[V][V]) {

printf("Edge \tWeight\n");

for (int i = 1; i < V; i++)

printf("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);

}

void primMST(int graph[V][V]) {

int parent[V];

int key[V];

bool mstSet[V];

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

key[i] = INT\_MAX;

mstSet[i] = false;

}

key[0] = 0;

parent[0] = -1;

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

int u = minKey(key, mstSet);

mstSet[u] = true;

for (int v = 0; v < V; v++)

if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v]) {

parent[v] = u;

key[v] = graph[u][v];

}

}

printMST(parent, graph);

}

int main() {

int graph[V][V] = {

{0, 2, 0, 6, 0},

{2, 0, 3, 8, 5},

{0, 3, 0, 0, 7},

{6, 8, 0, 0, 9},

{0, 5, 7, 9, 0}

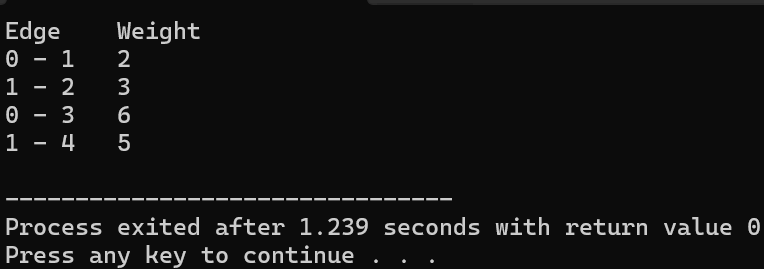
};

primMST(graph);

return 0;

}

**Output:**

****

1. Write a c-program to find out optimal binary search tree Using Dynamic programming.

**Code:**

#include<stdio.h>

void sorted(int array[], int n){

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

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

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

int temp=array[j];

array[j]=array[j+1];

array[j+1]=temp;

}

}

}

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

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

}

printf("]");

}

int binarysearch(int array[], int n,int target){

int low=0,high=n-1;

while(low<=high){

int mid=(low+high)/2;

if(array[mid]==target){

return mid;

}

else if(array[mid]>target){

high=mid-1;

}

else{

low=mid+1;

}

}

return -1;

}

int main(){

int n,target;

printf("enter no of elements in array:");

scanf("%d",&n);

int array[n],i,j;

printf("enter elements in array:");

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

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

}

printf("input array=[");

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

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

}

printf("]");

printf("\nenter target element:");

scanf("%d",&target);

printf("sorted array:[");

sorted(array,n);

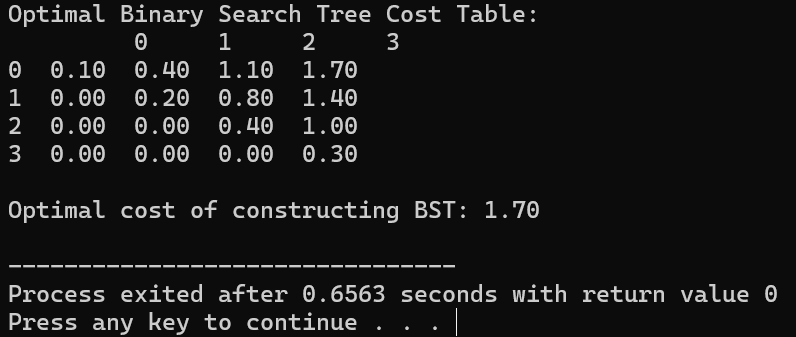
int result=binarysearch(array,n,target);

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

return 0;

}

**Output:**

****

1. Write a c-program to perform Binomial coefficient of a given number using dynamic programming.

**Code:**

#include <stdio.h>

int binomialCoeff(int n, int k) {

int C[n + 1][k + 1];

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

for (int j = 0; j <= k && j <= i; j++) {

if (j == 0 || j == i) {

C[i][j] = 1;

} else {

C[i][j] = C[i - 1][j - 1] + C[i - 1][j];

}

}

}

return C[n][k];

}

int main() {

int n, k;

printf("Enter n and k for C(n, k): ");

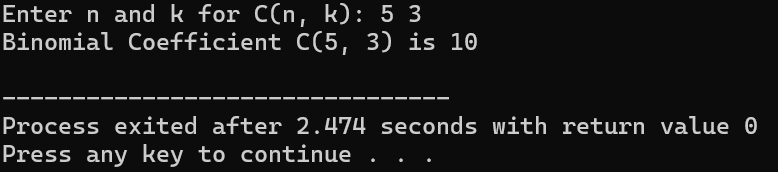
scanf("%d %d", &n, &k);

printf("Binomial Coefficient C(%d, %d) is %d\n", n, k, binomialCoeff(n, k));

return 0;

}

**Output:**

****

1. Write a c-program to find the reverse of a given number.

**Code:**

#include <stdio.h>

int reverseNumber(int num) {

int reversed = 0;

while (num != 0) {

int digit = num % 10;

reversed = reversed \* 10 + digit;

num /= 10;

}

return reversed;

}

int main() {

int num;

printf("Enter a number: ");

scanf("%d", &num);

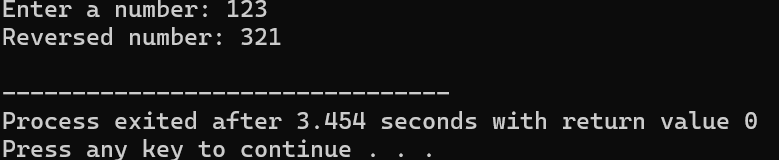
int reversed = reverseNumber(num);

printf("Reversed number: %d\n", reversed);

return 0;

}

**Output:**

****

1. Write a c-program to find a perfect number.

**Code:**

#include <stdio.h>

int isPerfectNumber(int num) {

int sum = 0;

for (int i = 1; i <= num / 2; i++) {

if (num % i == 0) {

sum += i;

}

}

if (sum == num) {

return 1;

} else {

return 0;

}

}

int main() {

int num;

printf("Enter a number: ");

scanf("%d", &num);

if (isPerfectNumber(num)) {

printf("%d is a perfect number.\n", num);

} else {

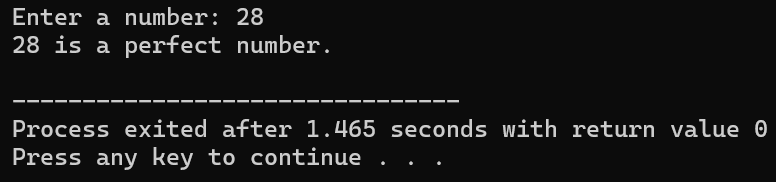
printf("%d is not a perfect number.\n", num);

}

return 0;

}

**Output:**

****

1. Write a c-program to perform TSP using dynamic programming.

**Code:**

#include <stdio.h>

#include <limits.h>

#define MAX 16

#define INF INT\_MAX

int dist[MAX][MAX];

int dp[1 << MAX][MAX];

int tsp(int n, int mask, int pos) {

if (mask == (1 << n) - 1) {

return dist[pos][0];

}

if (dp[mask][pos] != -1) {

return dp[mask][pos];

}

int ans = INF;

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

if ((mask & (1 << city)) == 0) {

int newAns = dist[pos][city] + tsp(n, mask | (1 << city), city);

ans = (ans < newAns) ? ans : newAns;

}

}

return dp[mask][pos] = ans;

}

int main() {

int n;

printf("Enter the number of cities: ");

scanf("%d", &n);

printf("Enter the distance matrix (adjacency matrix):\n");

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

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

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

}

}

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

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

dp[i][j] = -1;

}

}

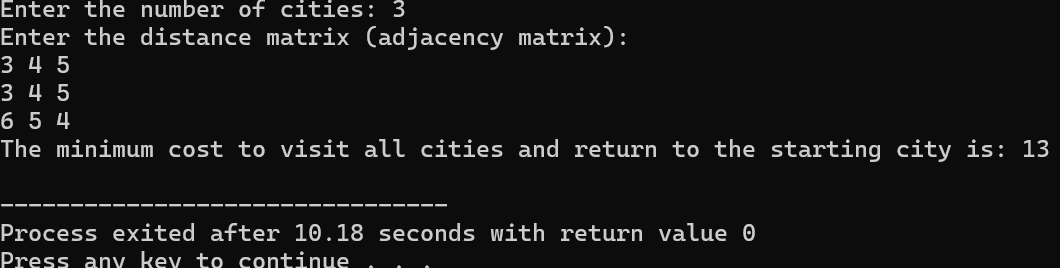
int result = tsp(n, 1, 0);

printf("The minimum cost to visit all cities and return to the starting city is: %d\n", result);

return 0;

}

**Output:**

****

1. Write a c-program to print the pattern for n=4.

**Code:**

#include <stdio.h>

int main() {

int n = 4;

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

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

printf(" ");

}

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

printf("%d ", j);

}

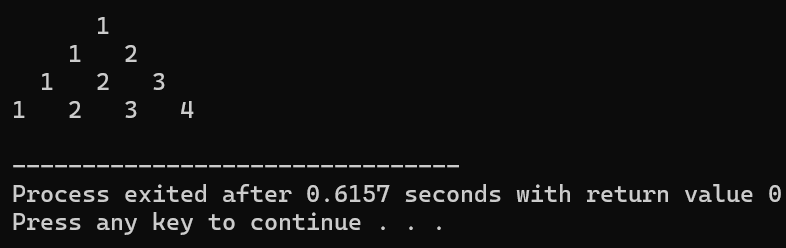
printf("\n");

}

return 0;

}

**Output:**

****

1. Write a c-program to perform Floyd’s algorithm.

**Code:**

#include <stdio.h>

#define INF 99999

void floydWarshall(int graph[5][5], int n) {

int dist[n][n];

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

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

dist[i][j] = graph[i][j];

}

}

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

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

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

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

if (dist[i][j] == INF) {

printf("INF\t");

} else {

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

}

}

printf("\n");

}

}

int main() {

int n = 5;

int graph[5][5] = {

{0, 6,INF,INF,INF},

{INF,0,5,INF,12},

{11,INF,0,INF,INF},

{INF,8,INF,0,INF},

{INF,INF,9,10,0}

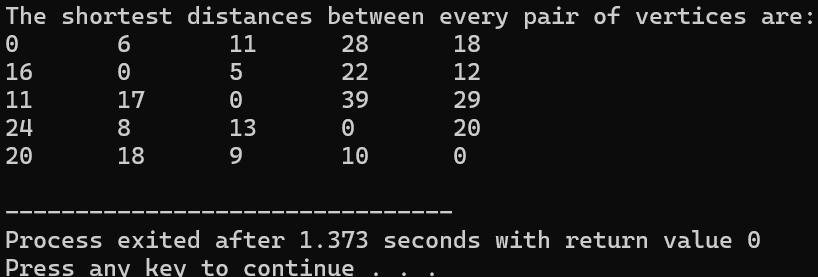
};

floydWarshall(graph, n);

return 0;

}

**Output:**

****

1. Write a c-program to print pascals triangle.

**Code:**

#include <stdio.h>

void generatePascalTriangle(int n) {

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

int value = 1;

for (int space = 1; space <= n - row; space++) {

printf(" ");

}

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

printf("%4d", value);

value = value \* (row - column) / (column + 1);

}

printf("\n");

}

}

int main() {

int n;

printf("Enter the number of rows for Pascal's Triangle: ");

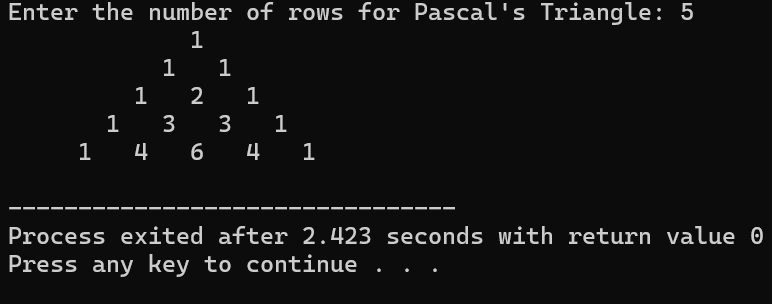
scanf("%d", &n);

generatePascalTriangle(n);

return 0;

}

**Output:**

****

1. Write a c-program to find sum of digits.

**Code:**

#include <stdio.h>

int sumOfDigits(int number) {

int sum = 0;

while (number > 0) {

int digit = number % 10;

sum += digit;

number /= 10;

}

return sum;

}

int main() {

int number;

printf("Enter a number: ");

scanf("%d", &number);

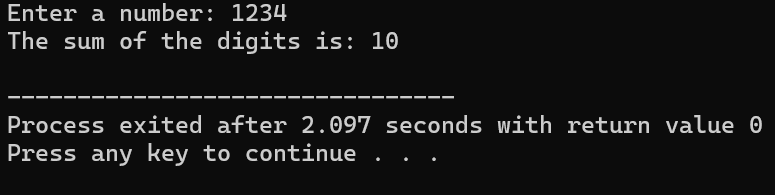
int result = sumOfDigits(number);

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

return 0;

}

**Output:**

****

1. Write a c-program to insert a new number in a list.

**Code:**

#include <stdio.h>

void insertNumber(int list[], int \*size, int number, int position) {

if (position < 1 || position > \*size + 1) {

printf("Invalid position!\n");

return;

}

for (int i = \*size; i >= position; i--) {

list[i] = list[i - 1];

}

list[position - 1] = number;

(\*size)++;

}

int main() {

int list[100], size, number, position;

printf("Enter the number of elements in the list: ");

scanf("%d", &size);

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

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

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

}

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

scanf("%d", &number);

printf("Enter the position to insert the number: ");

scanf("%d", &position);

insertNumber(list, &size, number, position);

printf("List after insertion:\n");

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

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

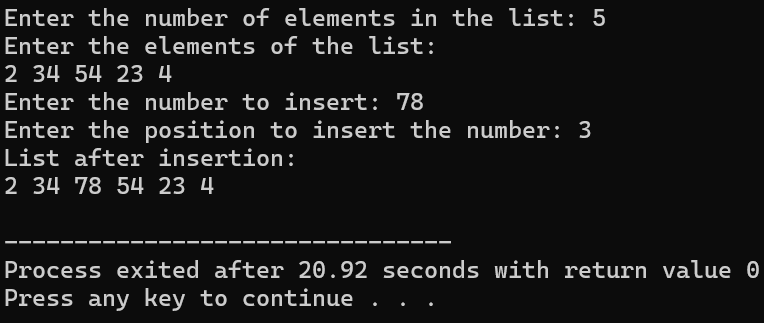
}

printf("\n");

return 0;

}

**Output:**

****

1. Write a c-program to perform Sum of subsets using backtracking.

**Code:**

#include <stdio.h>

int n, target;

int set[100], subset[100];

void sumOfSubsets(int index, int curr\_sum, int start) {

if (curr\_sum == target) {

printf("{ ");

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

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

}

printf("}\n");

return;

}

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

if (curr\_sum + set[i] <= target) {

subset[index] = set[i];

sumOfSubsets(index + 1, curr\_sum + set[i], i + 1);

}

}

}

int main() {

printf("Enter the number of elements in the set: ");

scanf("%d", &n);

printf("Enter the elements of the set: ");

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

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

}

printf("Enter the target sum: ");

scanf("%d", &target);

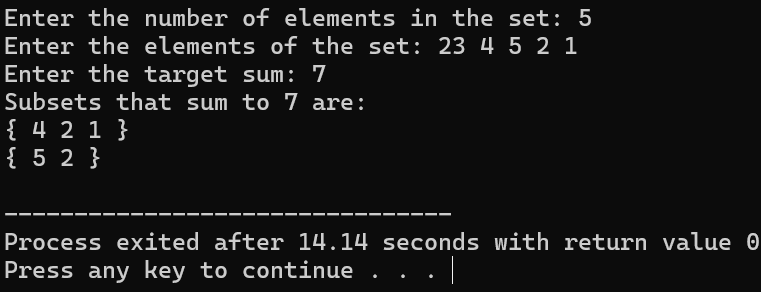
printf("Subsets that sum to %d are:\n", target);

sumOfSubsets(0, 0, 0);

return 0;

}

**Output:**

****

1. Write a c-program to perform Graph colouring using backtracking.

**Code:**

#include <stdio.h>

#include <stdbool.h>

#define MAX 10

int V;

int graph[MAX][MAX];

int colors[MAX];

bool isSafe(int v, int c) {

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

if (graph[v][i] && colors[i] == c) {

return false;

}

}

return true;

}

bool graphColoringUtil(int v, int m) {

if (v == V) {

return true;

}

for (int c = 1; c <= m; c++) {

if (isSafe(v, c)) {

colors[v] = c;

if (graphColoringUtil(v + 1, m)) {

return true;

}

colors[v] = 0;

}

}

return false;

}

bool graphColoring(int m) {

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

colors[i] = 0;

}

if (!graphColoringUtil(0, m)) {

printf("Solution does not exist.\n");

return false;

}

printf("Solution Exists: Following are the assigned colors:\n");

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

printf("Vertex %d -> Color %d\n", i, colors[i]);

}

return true;

}

int main() {

int m;

printf("Enter the number of vertices in the graph: ");

scanf("%d", &V);

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

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

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

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

}

}

printf("Enter the maximum number of colors: ");

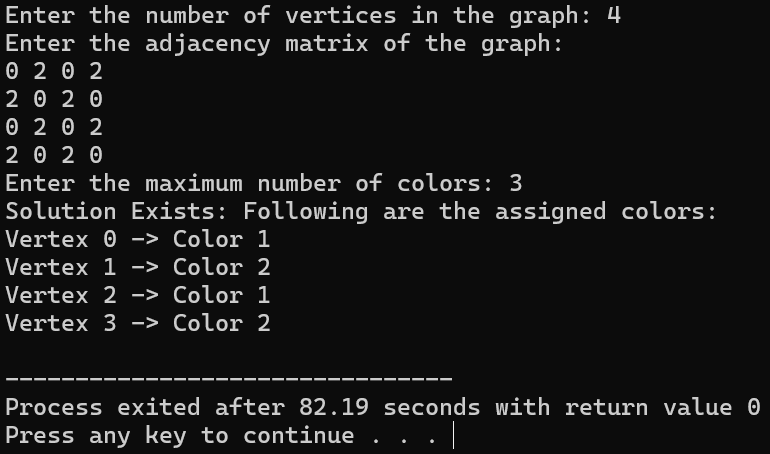
scanf("%d", &m);

graphColoring(m);

return 0;

}

**Output:**

****

1. Write a c-program to compute container loading problem.

**Code:**

#include <stdio.h>

int totalWeight = 0;

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

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void heapify(int arr[], int n, int i) {

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if (left < n && arr[left] > arr[largest]) {

largest = left;

}

if (right < n && arr[right] > arr[largest]) {

largest = right;

}

if (largest != i) {

swap(&arr[i], &arr[largest]);

heapify(arr, n, largest);

}

}

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

for (int i = n / 2 - 1; i >= 0; i--) {

heapify(arr, n, i);

}

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

swap(&arr[0], &arr[i]);

heapify(arr, i, 0);

}

}

int main() {

int n;

int capacity;

printf("Enter the number of items: ");

scanf("%d", &n);

int weights[n];

int excludedWeights[100];

int excludedCount = 0;

printf("Enter the weights of the items:\n");

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

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

}

printf("Enter the capacity of the container: ");

scanf("%d", &capacity);

heapSort(weights, n);

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

if (totalWeight + weights[i] <= capacity) {

totalWeight += weights[i];

} else {

excludedWeights[excludedCount++] = weights[i];

}

}

int remainingWeight = capacity - totalWeight;

printf("Remaining weight in the container: %d\n", remainingWeight);

printf("Excluded items (weights): ");

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

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

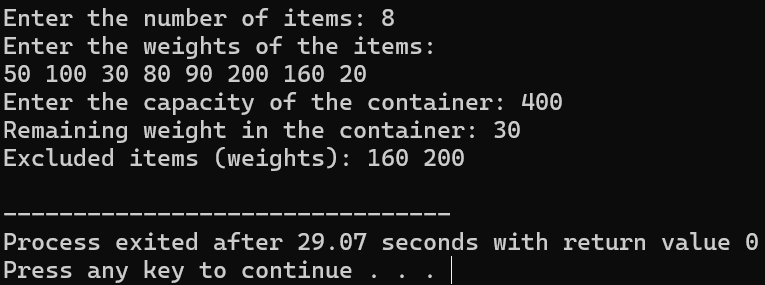
}

printf("\n");

return 0;

}

**Output:**

****

1. Write a c-program to generate the list of all factor for n value using recursion.

**Code:**

#include <stdio.h>

void findFactors(int n, int i) {

if (i > n) {

return;

}

if (n % i == 0) {

printf("%d ", i);

}

findFactors(n, i + 1);

}

int main() {

int n;

printf("Enter a number: ");

scanf("%d", &n);

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

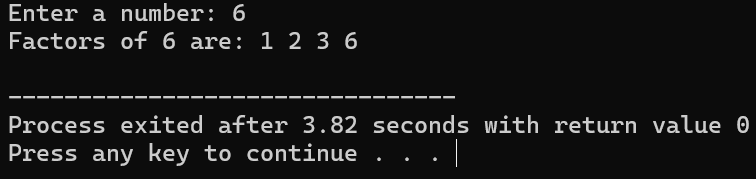
findFactors(n, 1);

printf("\n");

return 0;

}

**Output:**

****

1. Write a c-program to perform assignment problem using branch and bound.

**Code:**

**Output:**

1. Write a c-program to perform linear search.

**Code:**

#include<stdio.h>

int main(){

int n,i,target;

printf("enter no of elements:");

scanf("%d",&n);

int arr[10];

printf("enter elements:");

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

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

}

printf("enter target to search:");

scanf("%d",&target);

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

if(arr[i]==target){

printf("%d is found at index %d",target, i);

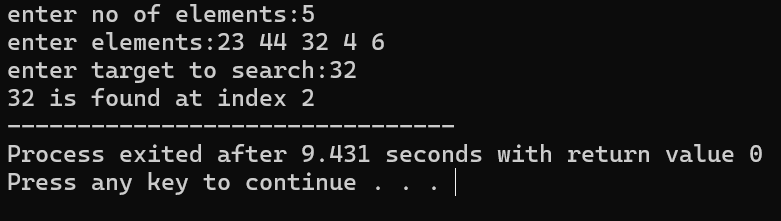
}

}

return 0;

}

**Output:**

****

1. Write a c-program to find all Hamiltonian circuit using backtracking.

**Code:**

#include <stdio.h>

#include <stdbool.h>

#define MAX\_VERTICES 10

int n, graph[MAX\_VERTICES][MAX\_VERTICES], path[MAX\_VERTICES];

bool isSafe(int v, int pos) {

if (graph[path[pos - 1]][v] == 0) {

return false;

}

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

if (path[i] == v) {

return false;

}

}

return true;

}

void hamiltonianCircuitUtil(int pos) {

if (pos == n) {

if (graph[path[pos - 1]][path[0]] == 1) {

printf("Hamiltonian Circuit: ");

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

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

}

printf("%d\n", path[0]);

}

return;

}

for (int v = 1; v < n; v++) {

if (isSafe(v, pos)) {

path[pos] = v;

hamiltonianCircuitUtil(pos + 1);

path[pos] = -1;

}

}

}

void findHamiltonianCircuits() {

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

path[i] = -1;

}

path[0] = 0;

hamiltonianCircuitUtil(1);

}

int main() {

printf("Enter the number of vertices: ");

scanf("%d", &n);

printf("Enter the adjacency matrix (0 or 1):\n");

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

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

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

}

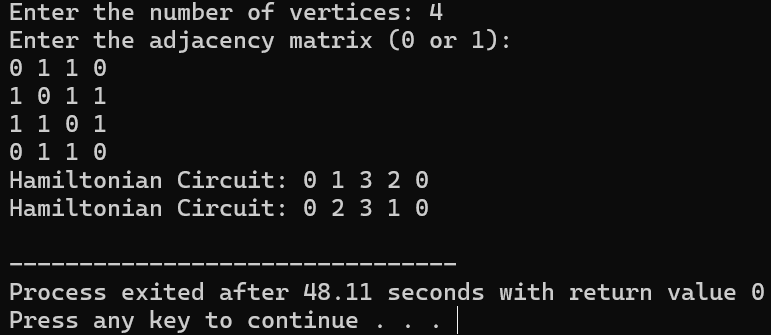
}

findHamiltonianCircuits();

return 0;

}

**Output:**

****

1. Write a c-program to perform n-queens problem using backtracking.

**Code:**

#include <stdio.h>

#include <stdbool.h>

#define MAX 10

int board[MAX][MAX], n;

void printBoard() {

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

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

if (board[i][j] == 1) {

printf("Q ");

} else {

printf(". ");

}

}

printf("\n");

}

printf("\n");

}

bool isSafe(int row, int col) {

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

if (board[row][i] == 1) {

return false;

}

}

for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {

if (board[i][j] == 1) {

return false;

}

}

for (int i = row, j = col; i < n && j >= 0; i++, j--) {

if (board[i][j] == 1) {

return false;

}

}

return true;

}

bool solveNQueens(int col) {

if (col >= n) {

return true;

}

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

if (isSafe(row, col)) {

board[row][col] = 1;

if (solveNQueens(col + 1)) {

return true;

}

board[row][col] = 0;

}

}

return false;

}

int main() {

printf("Enter the number of queens (n): ");

scanf("%d", &n);

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

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

board[i][j] = 0;

}

}

if (solveNQueens(0)) {

printf("Solution to the N-Queens problem:\n");

printBoard();

} else {

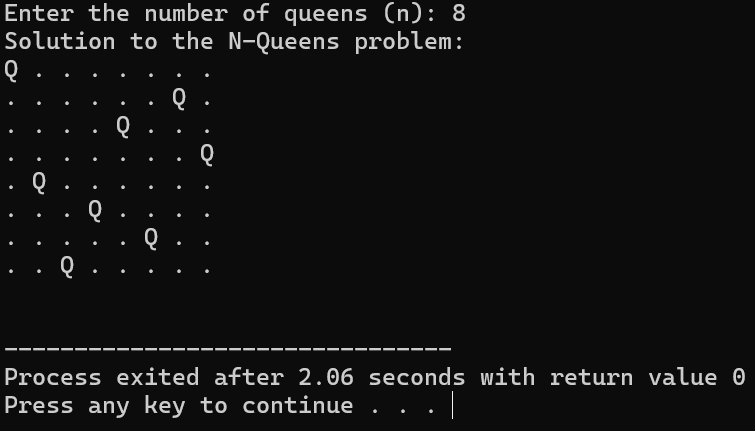
printf("Solution does not exist.\n");

}

return 0;

}

**Output:**

****

1. Write a c-program to find the optimal cost by using appropriate algorithm.

**Code:**

#include <stdio.h>

#include <limits.h>

#define MAX 10

int cost[MAX][MAX], n;

void subtractRowMin() {

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

int min = INT\_MAX;

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

if (cost[i][j] < min) {

min = cost[i][j];

}

}

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

cost[i][j] -= min;

}

}

}

void subtractColumnMin() {

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

int min = INT\_MAX;

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

if (cost[i][j] < min) {

min = cost[i][j];

}

}

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

cost[i][j] -= min;

}

}

}

void printMatrix() {

printf("Cost Matrix after Reduction:\n");

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

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

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

}

printf("\n");

}

}

int main() {

printf("Enter the number of workers/tasks (n): ");

scanf("%d", &n);

printf("Enter the cost matrix (n x n):\n");

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

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

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

}

}

subtractRowMin();

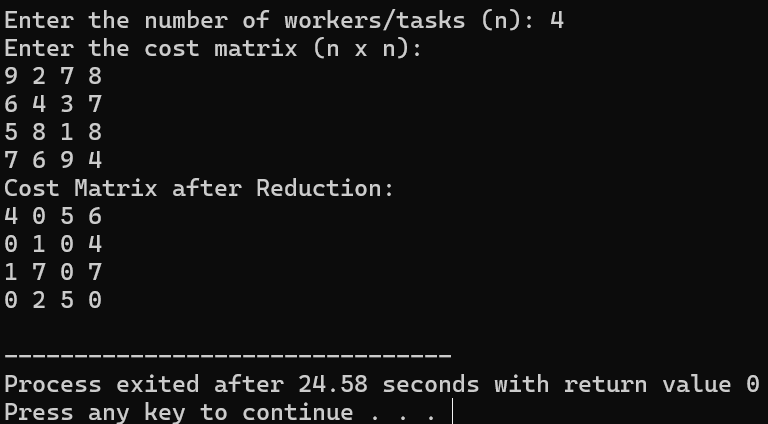
subtractColumnMin();

printMatrix();

return 0;

}

**Output:**

****

1. Write a c-program to print minimum and maximum value sequency for all the numbers in a list.

**Code:**

#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 Value: %d\n", min);

printf("Maximum Value: %d\n", max);

}

int main() {

int n;

printf("Enter the number of elements in the list: ");

scanf("%d", &n);

int arr[n];

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

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

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

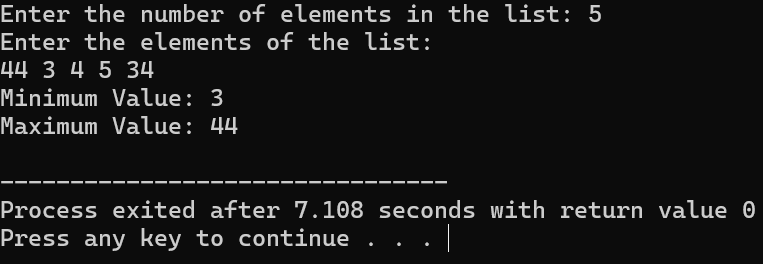
}

findMinMax(arr, n);

return 0;

}

**Output:**

****