1.Write a program to Print Fibonacci Series using recursion.

#include <stdio.h>

// Function to calculate the nth Fibonacci number using recursion

int fibonacci(int n) {

if (n <= 1)

return n;

else

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

}

int main() {

int n, i;

// Input: Number of terms to be printed

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

scanf("%d", &n);

// Output: Fibonacci series

printf("Fibonacci Series: ");

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

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

}

return 0;

}



2.Write a program to check the given no is Armstrong or not.

#include <stdio.h>

#include <math.h>

// Function to count the number of digits in a number

int countDigits(int num) {

int count = 0;

while (num != 0) {

num /= 10;

count++;

}

return count;

}

// Function to check if a number is an Armstrong number

int isArmstrong(int num) {

int sum = 0, temp, remainder, n = 0;

// Calculate the number of digits

n = countDigits(num);

temp = num;

while (temp != 0) {

remainder = temp % 10;

sum += pow(remainder, n);

temp /= 10;

}

if (sum == num)

return 1;

else

return 0;

}

int main() {

int num;

// Input: Number to be checked

printf("Enter a number: ");

scanf("%d", &num);

// Output: Check if the number is an Armstrong number

if (isArmstrong(num))

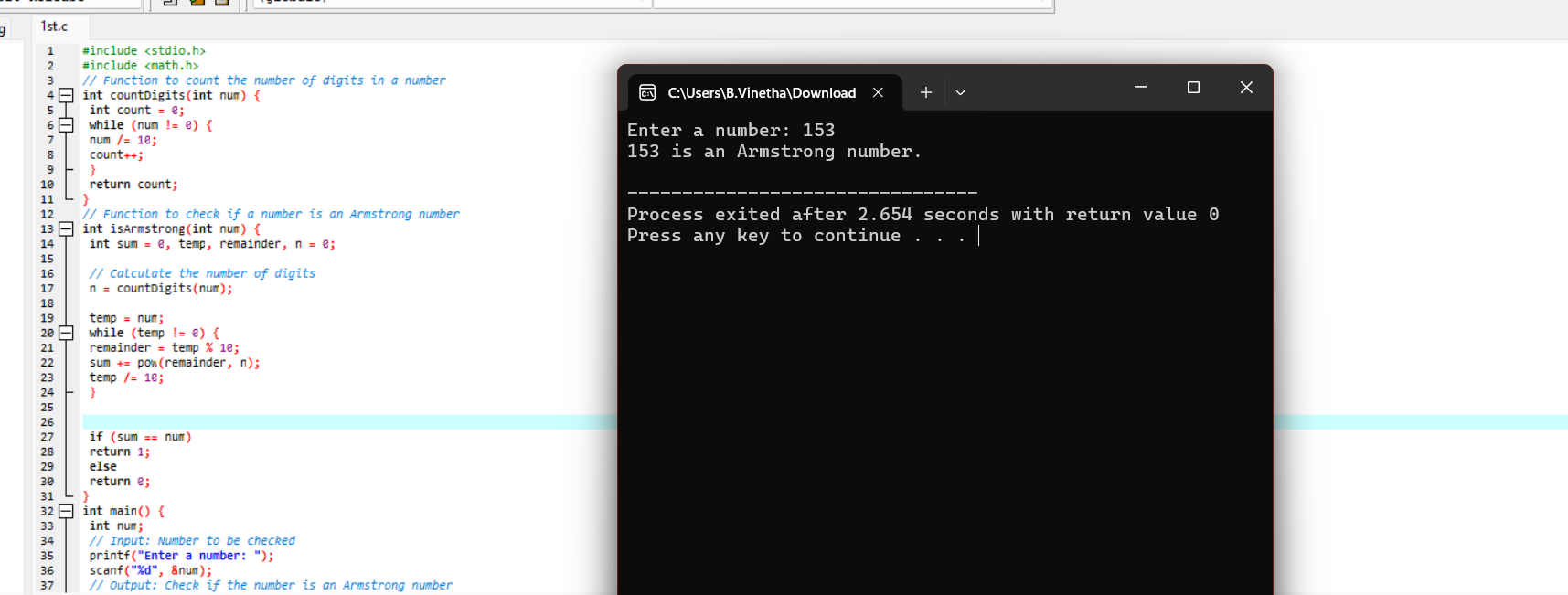
printf("%d is an Armstrong number.\n", num);

else

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

return 0;

}



3.Program to find the GCD of two numbers .

#include <stdio.h>

// Function to calculate GCD using the Euclidean algorithm

int gcd(int a, int b) {

if (b == 0)

return a;

else

return gcd(b, a % b);

}

int main() {

int num1, num2, result;

// Input: Two numbers

printf("Enter two integers: ");

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

// Calculate GCD

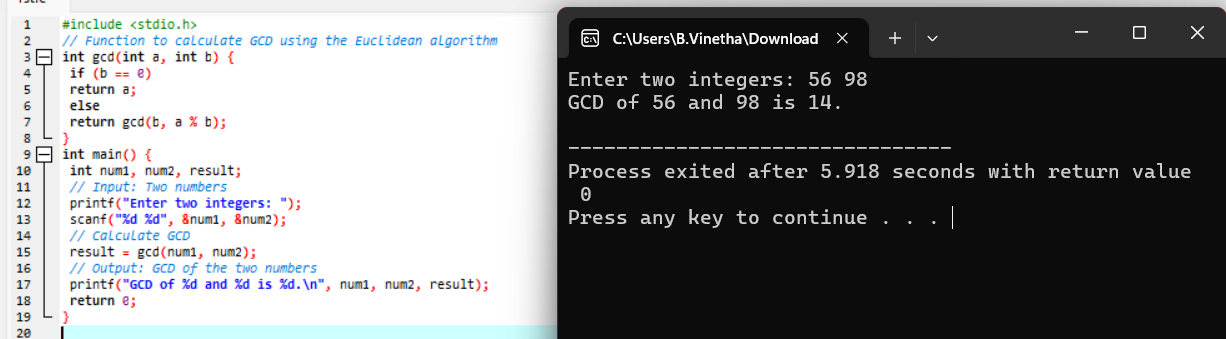
result = gcd(num1, num2);

// Output: GCD of the two numbers

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

return 0;

}



4. Write a program to get the largest element of an array.

#include <stdio.h>

int main() {

int n, i;

int largest;

// Input: Number of elements in the array

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

scanf("%d", &n);

// Declare an array of size n

int arr[n];

// Input: Elements of the array

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

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

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

}

// Initialize the largest element with the first element

largest = arr[0];

// Iterate through the array to find the largest element

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

if (arr[i] > largest) {

largest = arr[i];

}

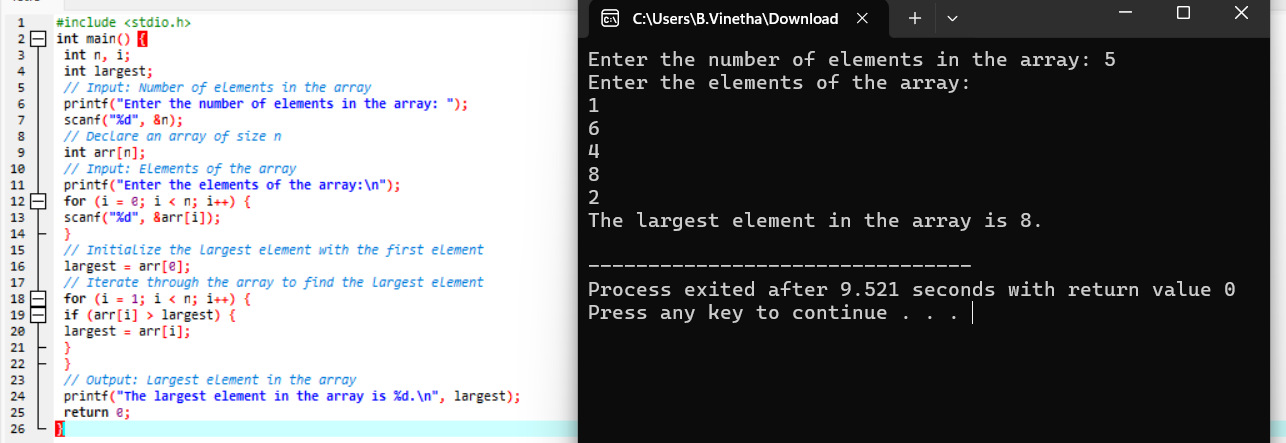
}

// Output: Largest element in the array

printf("The largest element in the array is %d.\n", largest);

return 0;

}



5. Write a program to find the Factorial of a number

#include <stdio.h>

int factorial(int n) {

if(n == 0)

return 1;

else

return n \* factorial(n - 1);

}

int main() {

int num;

printf("Enter a number: ");

scanf("%d", &num);

if(num < 0)

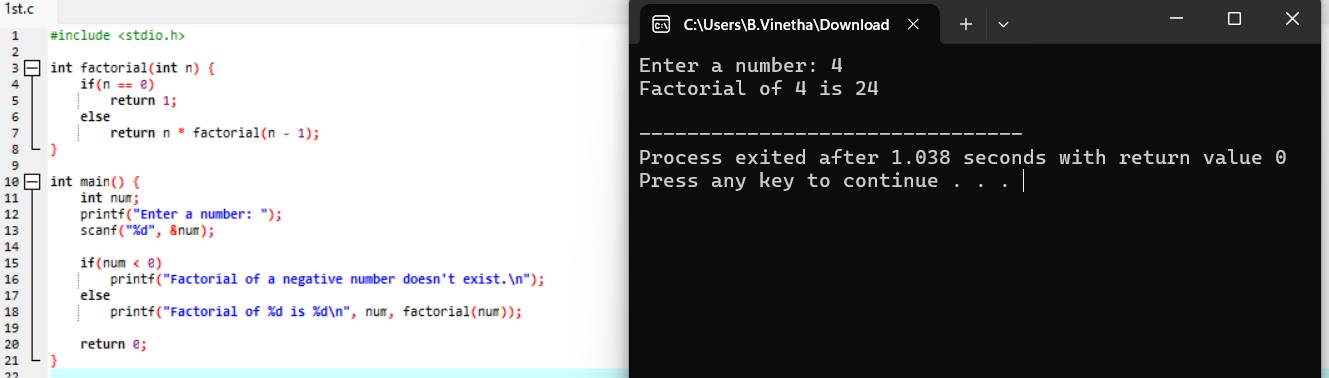
printf("Factorial of a negative number doesn't exist.\n");

else

printf("Factorial of %d is %d\n", num, factorial(num));

return 0;

}



6.Write a program to check a number is a prime number or not .

#include<stdio.h>

int main(){

int n,i,count=0;

printf("enter a number:");

scanf("%d",&n);

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

if(n%i==0){

count++;

}

}

if(count==2)

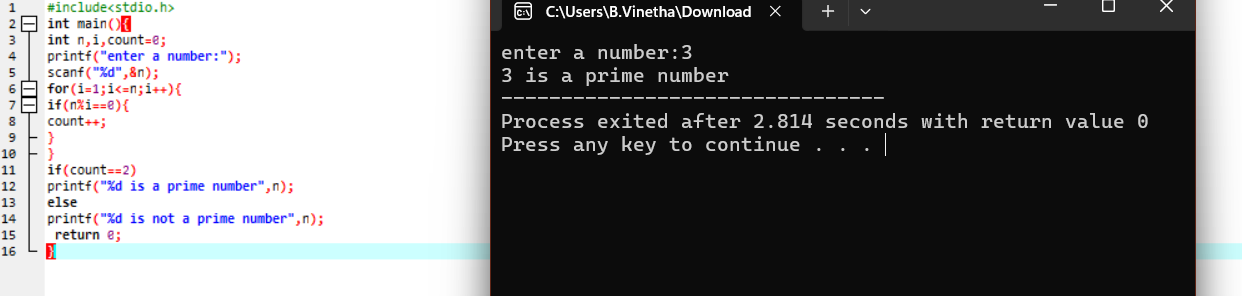
printf("%d is a prime number",n);

else

printf("%d is not a prime number",n);

return 0;

}



7.Write a program to perform Selection sort.

#include <stdio.h>

// Function to perform selection sort

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

int i, j, min\_idx, temp;

// Iterate over the array

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

// Assume the minimum element is the first element of the unsorted part

min\_idx = i;

// Find the actual minimum element in the unsorted part

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

if (arr[j] < arr[min\_idx]) {

min\_idx = j;

}

}

// Swap the found minimum element with the first element of the unsorted part

temp = arr[min\_idx];

arr[min\_idx] = arr[i];

arr[i] = temp;

}

}

// Function to print an array

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

int i;

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

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

}

printf("\n");

}

int main() {

int n, i;

// Input: Number of elements in the array

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

scanf("%d", &n);

// Declare an array of size n

int arr[n];

// Input: Elements of the array

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

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

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

}

// Perform selection sort

selectionSort(arr, n);

// Output: Sorted array

printf("Sorted array: \n");

printArray(arr, n);

return 0;

}



8.Write a program to perform Bubble sort

#include <stdio.h>

// Function to perform bubble sort

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

int i, j, temp;

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

// Last i elements are already in place

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

// Compare and swap if elements are in the wrong order

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

temp = arr[j];

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

arr[j+1] = temp;

}

}

}

}

// Function to print an array

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

int i;

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

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

}

printf("\n");

}

int main() {

int n, i;

// Input: Number of elements in the array

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

scanf("%d", &n);

// Declare an array of size n

int arr[n];

// Input: Elements of the array

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

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

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

}

// Perform bubble sort

bubbleSort(arr, n);

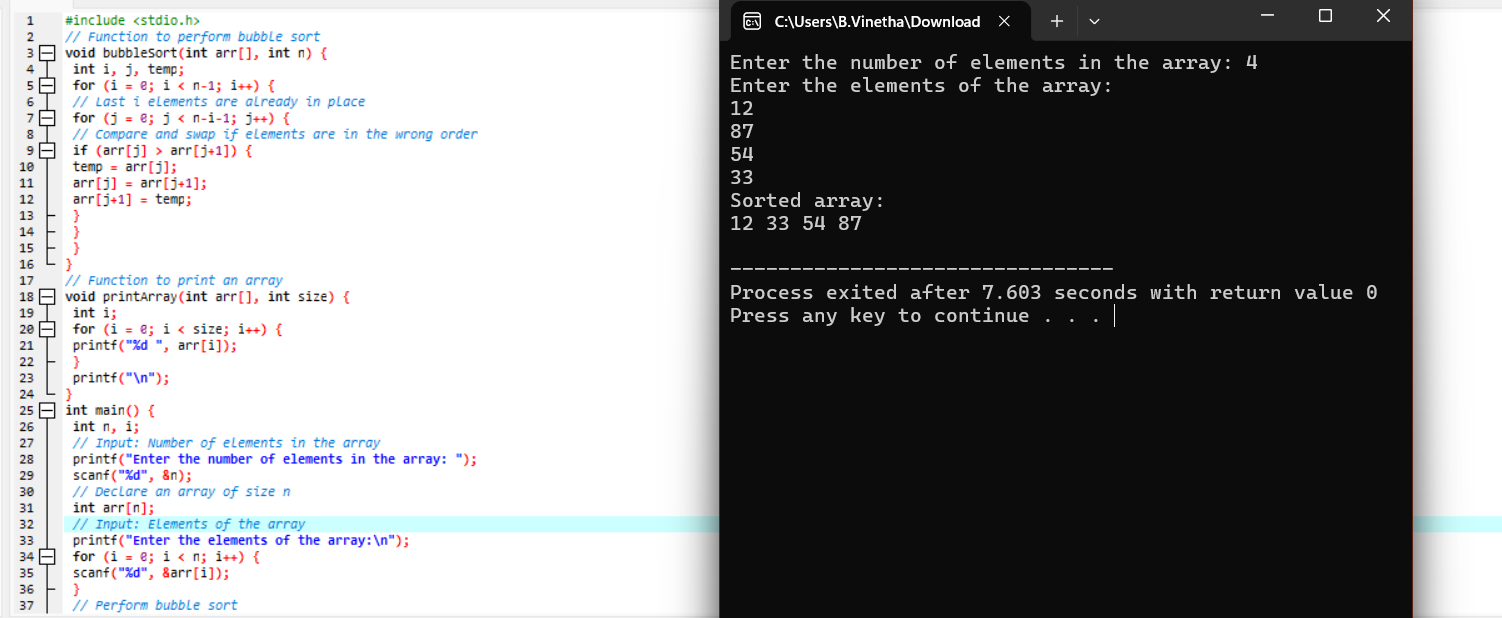
// Output: Sorted array

printf("Sorted array: \n");

printArray(arr, n);

return 0;

}



9.Write a program for to multiply two Matrix

#include <stdio.h>

// Function to multiply two matrices

void multiplyMatrices(int firstMatrix[][10], int secondMatrix[][10], int

resultMatrix[][10], int r1, int c1, int r2, int c2) {

int i, j, k;

// Initialize the result matrix to 0

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

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

resultMatrix[i][j] = 0;

}

}

// Multiply the matrices

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

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

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

resultMatrix[i][j] += firstMatrix[i][k] \*

secondMatrix[k][j];

}

}

}

}

// Function to print a matrix

void printMatrix(int matrix[][10], int row, int col) {

int i, j;

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

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

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

}

printf("\n");

}

}

int main() {

int r1, c1, r2, c2, i, j;

// Input: Dimensions of the first matrix

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

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

// Input: Dimensions of the second matrix

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

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

// Ensure the matrices can be multiplied

if (c1 != r2) {

printf("Error! Column of the first matrix must be equal to row of the second matrix.\n");

return -1;

}

// Declare the matrices

int firstMatrix[10][10], secondMatrix[10][10], resultMatrix[10][10];

// Input: Elements of the first matrix

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

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

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

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

}

}

// Input: Elements of the second matrix

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

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

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

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

}

}

// Multiply the matrices

multiplyMatrices(firstMatrix, secondMatrix, resultMatrix, r1, c1, r2,

c2);

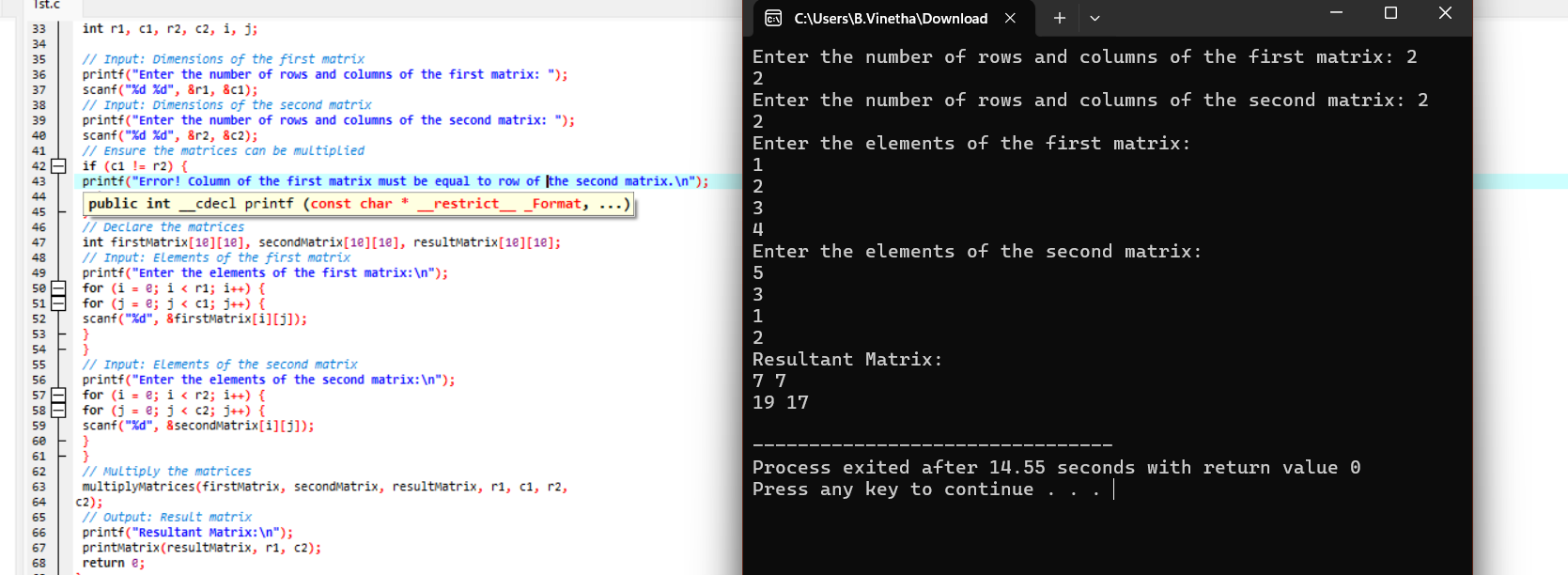
// Output: Result matrix

printf("Resultant Matrix:\n");

printMatrix(resultMatrix, r1, c2);

return 0;

}



10.Write a program for to check whether a given String is Palindrome or not

#include <stdio.h>

#include <string.h>

#include <stdbool.h>

// Function to check if a string is a palindrome

bool isPalindrome(char str[]) {

int left = 0;

int right = strlen(str) - 1;

while (left < right) {

if (str[left] != str[right]) {

return false;

}

left++;

right--;

}

return true;

}

int main() {

char str[100];

// Input: String to be checked

printf("Enter a string: ");

scanf("%s", str);

// Output: Check if the string is a palindrome

if (isPalindrome(str)) {

printf("\"%s\" is a palindrome.\n", str);

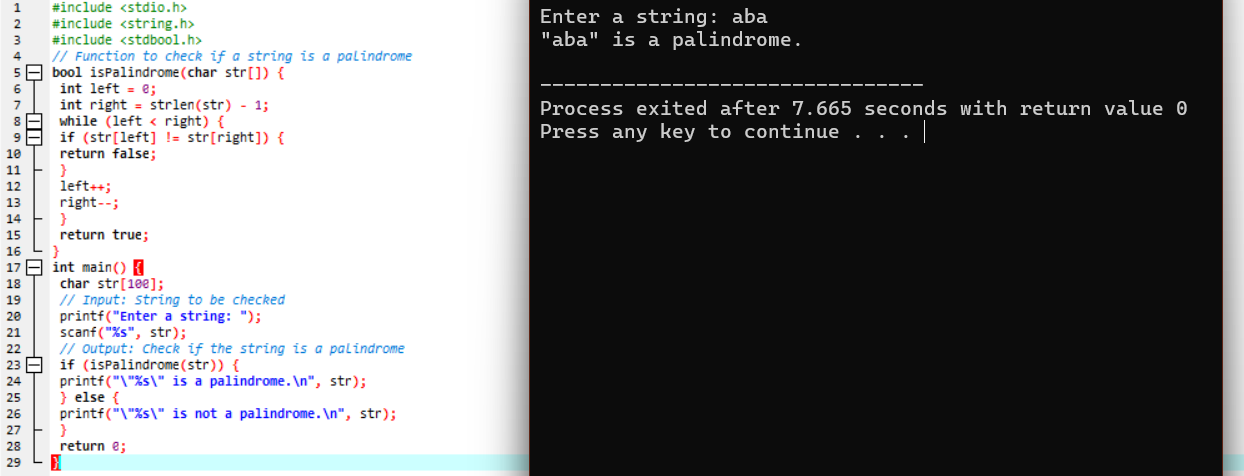
} else {

printf("\"%s\" is not a palindrome.\n", str);

}

return 0;

}



11.Write a program for to copy one string to another

#include <stdio.h>

#include <string.h>

int main() {

char source[100], destination[100];

// Input: Source string

printf("Enter the source string: ");

fgets(source, sizeof(source), stdin);

// Remove the newline character if present

source[strcspn(source, "\n")] = '\0';

// Copy the string using strcpy

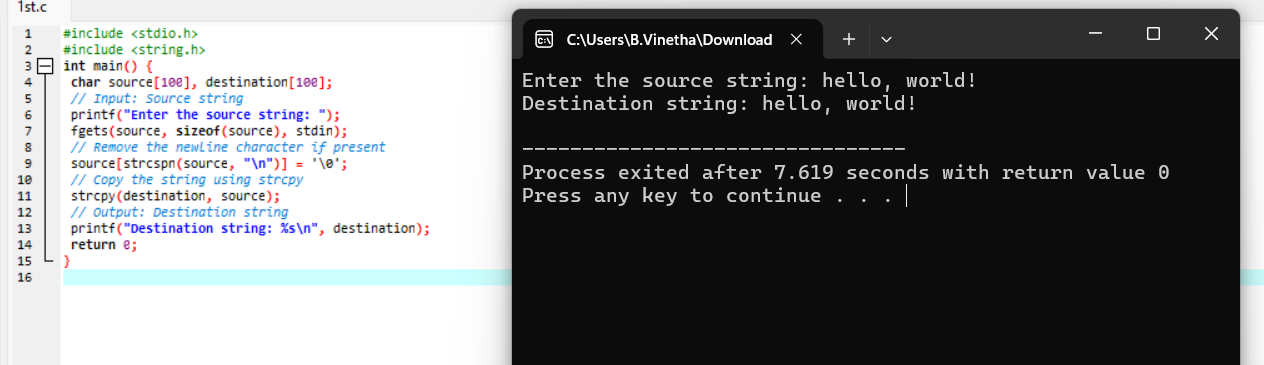
strcpy(destination, source);

// Output: Destination string

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

return 0;

}



12.Write a Program to perform binary search.

#include <stdio.h>

// Function to perform binary search

int binarySearch(int arr[], int size, int target) {

int low = 0;

int i;

int high = size - 1;

int mid;

while (low <= high) {

mid = low + (high - low) / 2;

// Check if target is present at mid

if (arr[mid] == target) {

return mid; // Target found

}

// If target is smaller, ignore the right half

if (arr[mid] > target) {

high = mid - 1;

}

// If target is larger, ignore the left half

else {

low = mid + 1;

}

}

return -1; // Target not found

}

int main() {

int n, target, result,i;

// Input: Number of elements

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

scanf("%d", &n);

// Declare an array of size n

int arr[n];

// Input: Elements of the array

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

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

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

}

// Input: Target value to search

printf("Enter the target value to search: ");

scanf("%d", &target);

// Perform binary search

result = binarySearch(arr, n, target);

// Output: Result of binary search

if (result != -1) {

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

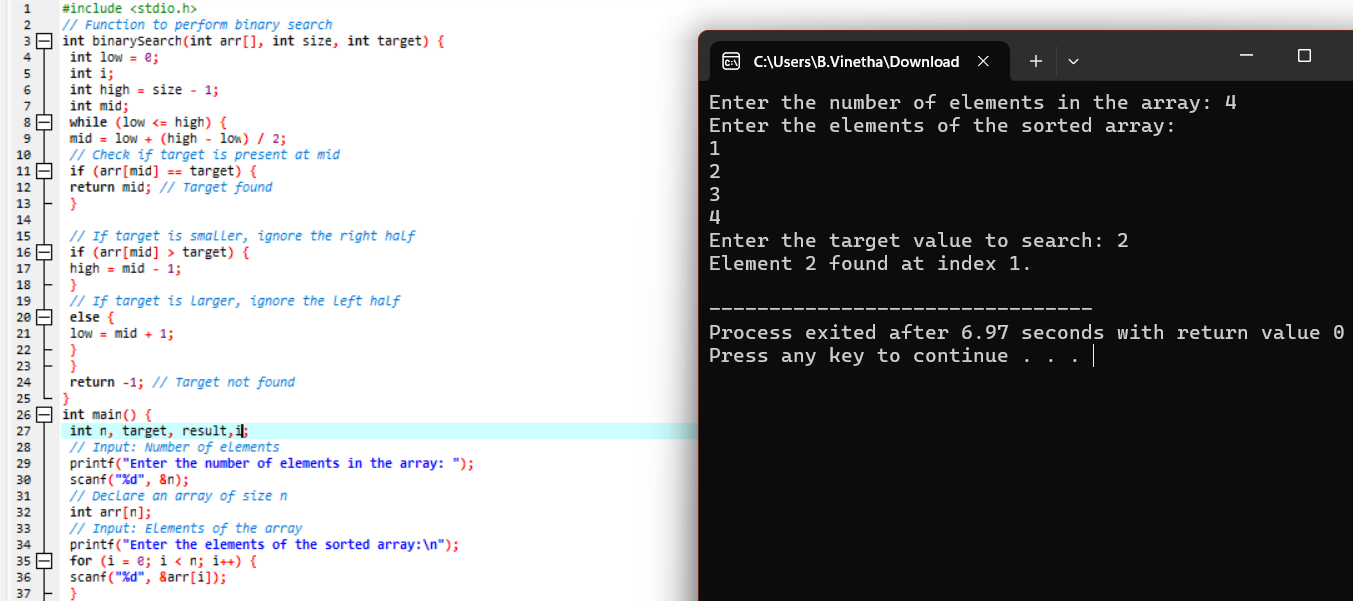
} else {

printf("Element %d not found in the array.\n", target);

}

return 0;

}



13.Write a program to print the reverse of a string

#include <stdio.h>

#include <string.h>

// Function to print the reverse of a string

void printReverse(char str[]) {

int i=0;

int length = strlen(str);

for ( i = length - 1; i >= 0; i--) {

printf("%c", str[i]);

}

printf("\n");

}

int main() {

char str[100];

// Input: String to be reversed

printf("Enter a string: ");

fgets(str, sizeof(str), stdin);

// Remove the newline character if present

str[strcspn(str, "\n")] = '\0';

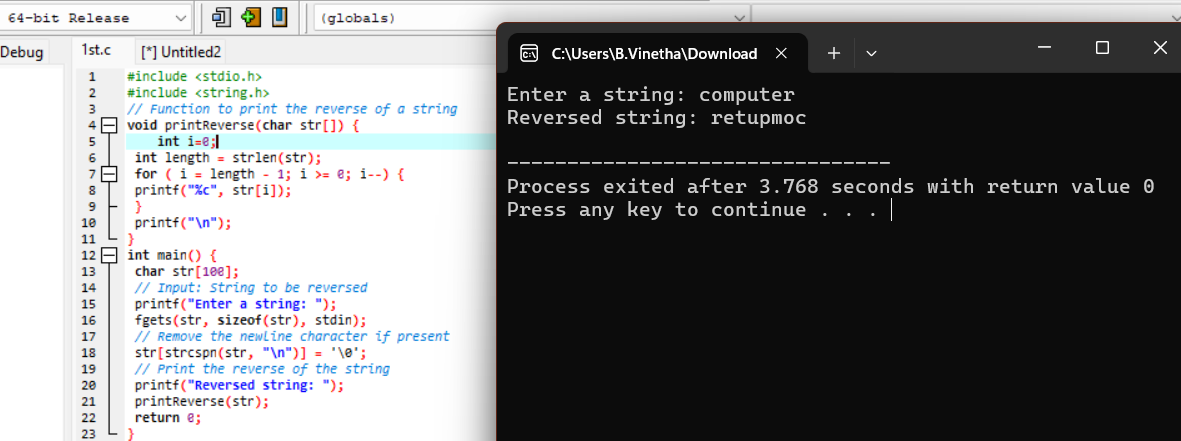
// Print the reverse of the string

printf("Reversed string: ");

printReverse(str);

return 0;

}



14. Write a program to find the length of a string.

#include <stdio.h>

#include <string.h>

int main() {

char str[100];

// Input: String from the user

printf("Enter a string: ");

fgets(str, sizeof(str), stdin);

// Remove the newline character if present

str[strcspn(str, "\n")] = '\0';

// Calculate the length of the string using strlen

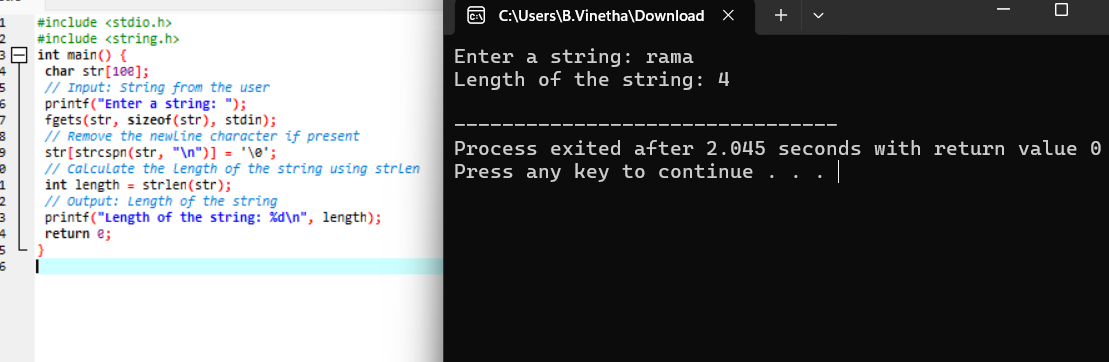
int length = strlen(str);

// Output: Length of the string

printf("Length of the string: %d\n", length);

return 0;

}



15.Write a program to perform Strassen’s Matrix Multiplication.

#include <stdio.h>

void add(int A[2][2], int B[2][2], int C[2][2]) {

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

C[i][j] = A[i][j] + B[i][j];

}

}

}

void subtract(int A[2][2], int B[2][2], int C[2][2]) {

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

C[i][j] = A[i][j] - B[i][j];

}

}

}

void strassen(int A[2][2], int B[2][2], int C[2][2]) {

int M1, M2, M3, M4, M5, M6, M7;

int temp1[2][2], temp2[2][2];

// Calculating M1 to M7

M1 = (A[0][0] + A[1][1]) \* (B[0][0] + B[1][1]);

M2 = (A[1][0] + A[1][1]) \* B[0][0];

M3 = A[0][0] \* (B[0][1] - B[1][1]);

M4 = A[1][1] \* (B[1][0] - B[0][0]);

M5 = (A[0][0] + A[0][1]) \* B[1][1];

M6 = (A[1][0] - A[0][0]) \* (B[0][0] + B[0][1]);

M7 = (A[0][1] - A[1][1]) \* (B[1][0] + B[1][1]);

// C11 = M1 + M4 - M5 + M7

C[0][0] = M1 + M4 - M5 + M7;

// C12 = M3 + M5

C[0][1] = M3 + M5;

// C21 = M2 + M4

C[1][0] = M2 + M4;

// C22 = M1 - M2 + M3 + M6

C[1][1] = M1 - M2 + M3 + M6;

}

int main() {

int A[2][2], B[2][2], C[2][2];

// Input matrices

printf("Enter the elements of 2x2 matrix A:\n");

for (int i = 0; i < 2; i++)

for (int j = 0; j < 2; j++)

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

printf("Enter the elements of 2x2 matrix B:\n");

for (int i = 0; i < 2; i++)

for (int j = 0; j < 2; j++)

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

// Perform Strassen's matrix multiplication

strassen(A, B, C);

// Output the result

printf("The resulting matrix C is:\n");

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

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

}

printf("\n");

}

return 0;

}



16.Write a program to perform Merge Sort.

#include <stdio.h>

#include <stdlib.h>

int i=0,j=0;

// Function to merge two halves

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

int n1 = mid - left + 1;

int n2 = right - mid;

// Create temporary arrays

int\* L = (int\*)malloc(n1 \* sizeof(int));

int\* R = (int\*)malloc(n2 \* sizeof(int));

// Copy data to temporary arrays L[] and R[]

for ( i = 0; i < n1; i++) {

L[i] = arr[left + i];

}

for ( j = 0; j < n2; j++) {

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

}

// Merge the temporary arrays back into arr[left..right]

int i = 0; // Initial index of the first subarray

int j = 0; // Initial index of the second subarray

int k = left; // Initial index of the merged subarray

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

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

} else {

arr[k++] = R[j++];

}

}

// Copy the remaining elements of L[], if any

while (i < n1) {

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

}

// Copy the remaining elements of R[], if any

while (j < n2) {

arr[k++] = R[j++];

}

// Free allocated memory

free(L);

free(R);

}

// Function to implement Merge Sort

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

if (left < right) {

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

// Recursively sort the first and second halves

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

// Merge the sorted halves

merge(arr, left, mid, right);

}

}

// Function to print an array

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

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

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

}

printf("\n");

}

int main() {

int arr[] = {12, 11, 13, 5, 6, 7};

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

printf("Given array:\n");

printArray(arr, size);

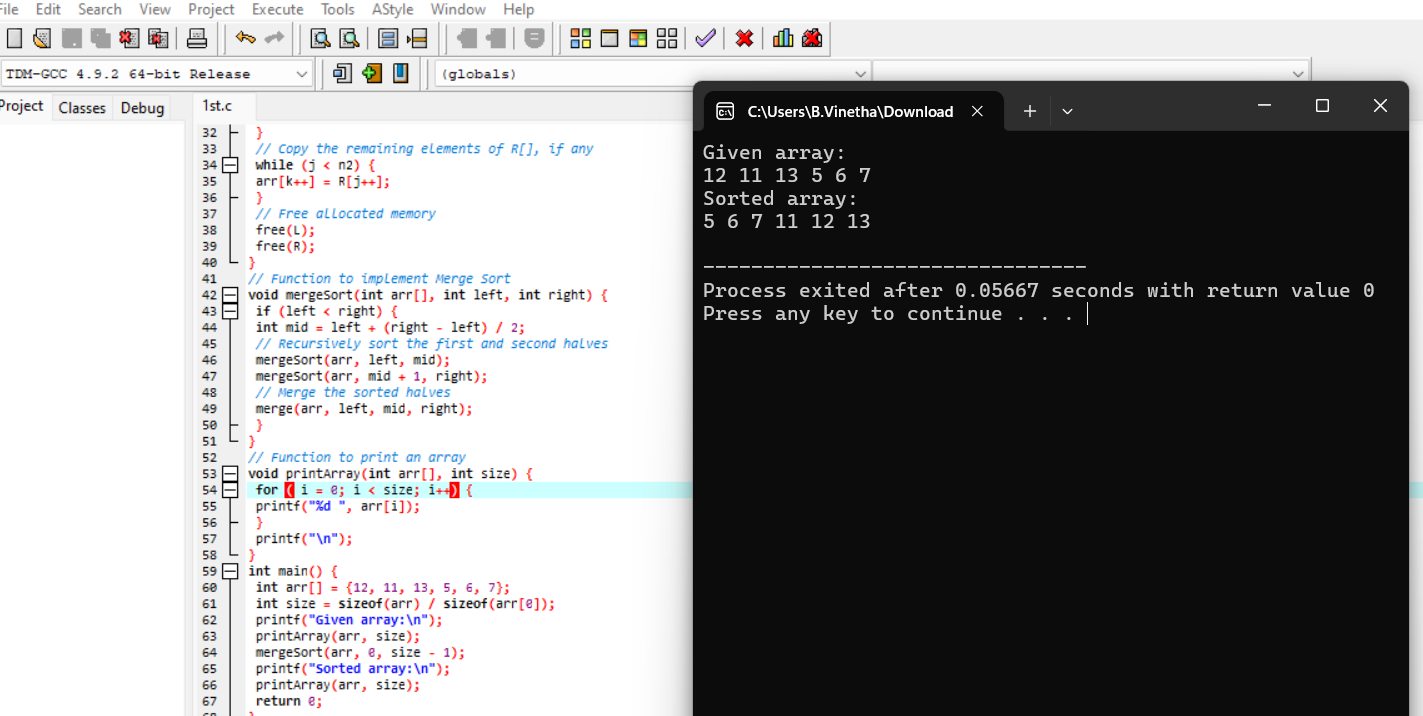
mergeSort(arr, 0, size - 1);

printf("Sorted array:\n");

printArray(arr, size);

return 0;

}



17.Using Divide and Conquer strategy to find Max and Min value in the list.

#include <stdio.h>

#include <limits.h>

// Function to find maximum and minimum using divide and conquer

void findMaxMin(int arr[], int left, int right, int \*max, int \*min) {

if (left == right) {

// Base case: only one element

\*max = arr[left];

\*min = arr[left];

} else if (right == left + 1) {

// Base case: two elements

if (arr[left] > arr[right]) {

\*max = arr[left];

\*min = arr[right];

} else {

\*max = arr[right];

\*min = arr[left];

}

} else {

// Divide

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

int leftMax, leftMin, rightMax, rightMin;

// Conquer

findMaxMin(arr, left, mid, &leftMax, &leftMin);

findMaxMin(arr, mid + 1, right, &rightMax, &rightMin);

// Combine

\*max = (leftMax > rightMax) ? leftMax : rightMax;

\*min = (leftMin < rightMin) ? leftMin : rightMin;

}

}

int main() {

int arr[] = {3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5};

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

int max, min;

// Find maximum and minimum

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

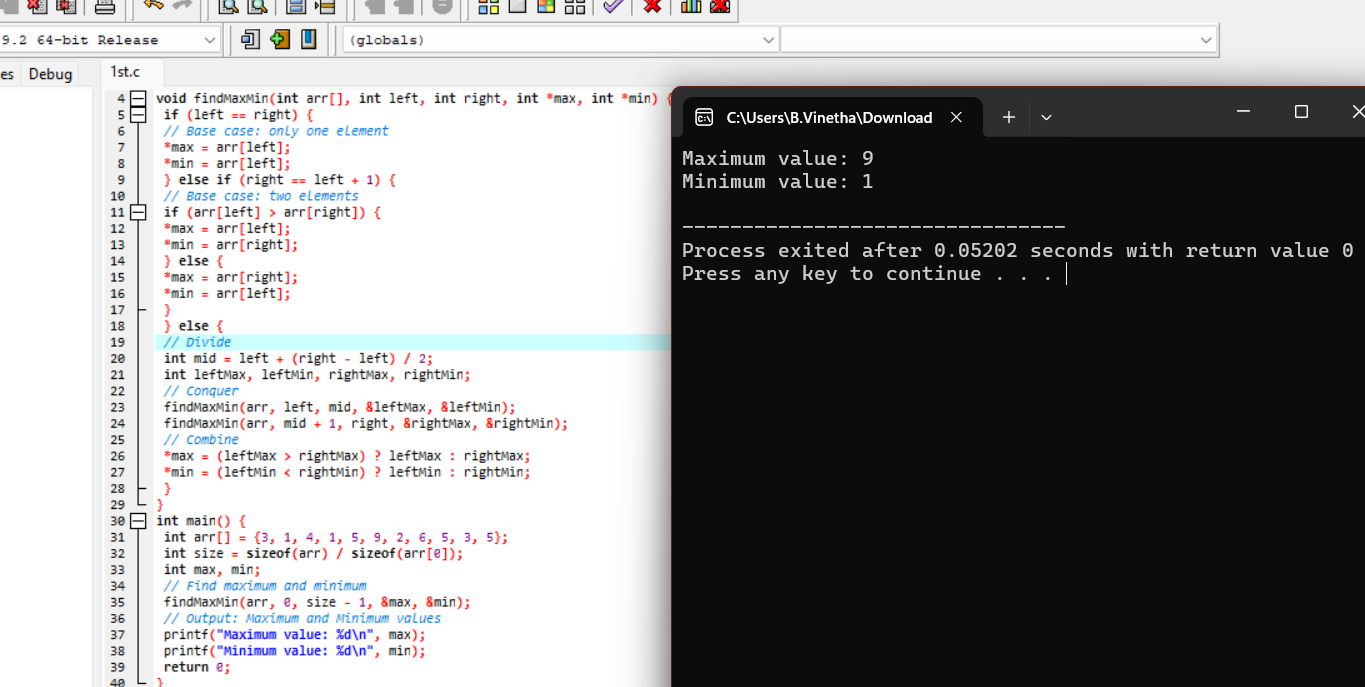
// Output: Maximum and Minimum values

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

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

return 0;

}



18.Write a program to generate all the prime numbers.

#include <stdio.h>

#include <stdbool.h>

int i=0,p=0;

// Function to generate prime numbers using Sieve of Eratosthenes

void sieveOfEratosthenes(int n) {

// Create a boolean array and initialize all entries as true

bool prime[n + 1];

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

prime[i] = true;

}

// 0 and 1 are not prime numbers

prime[0] = prime[1] = false;

for (p = 2; p \* p <= n; p++) {

// If prime[p] is not changed, then it is a prime

if (prime[p] == true) {

// Update all multiples of p

for (i = p \* p; i <= n; i += p) {

prime[i] = false;

}

}

}

// Print all prime numbers

printf("Prime numbers up to %d:\n", n);

for ( p = 2; p <= n; p++) {

if (prime[p]) {

printf("%d ", p);

}

}

printf("\n");

}

int main() {

int n;

// Input: Upper limit to generate prime numbers

printf("Enter the upper limit to generate prime numbers: ");

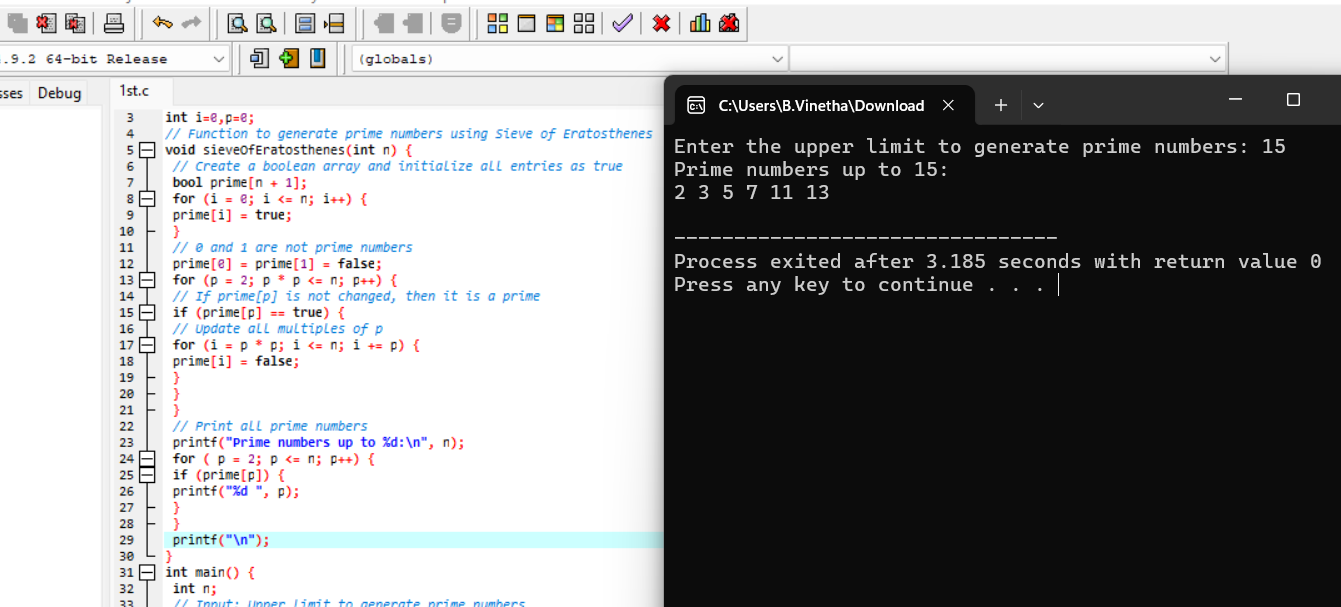
scanf("%d", &n);

// Generate and print prime numbers up to n

sieveOfEratosthenes(n);

return 0;

}



19. Write a program to perform Knapsack problem using greedy techniques.

#include <stdio.h>

int i=0;

// Structure to represent an item

typedef struct {

int value;

int weight;

float ratio;

} Item;

// Function to compare items based on their value-to-weight ratio

int compare(const void \*a, const void \*b) {

Item \*item1 = (Item \*)a;

Item \*item2 = (Item \*)b;

return (item2->ratio > item1->ratio) - (item2->ratio < item1->ratio);

}

// Function to perform the fractional knapsack problem

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

// Sort items by value-to-weight ratio

qsort(items, n, sizeof(Item), compare);

int currentWeight = 0;

float totalValue = 0.0;

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

if (currentWeight + items[i].weight <= capacity) {

// Take the whole item

currentWeight += items[i].weight;

totalValue += items[i].value;

} else {

// Take the fractional part of the item

int remaining = capacity - currentWeight;

totalValue += items[i].value \* ((float)remaining /

items[i].weight);

break;

}

}

return totalValue;

}

int main() {

int n, capacity;

// Input: Number of items

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

scanf("%d", &n);

Item items[n];

// Input: Value, Weight, and Capacity

printf("Enter the value and weight for each item:\n");

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

printf("Item %d - Value: ", i + 1);

scanf("%d", &items[i].value);

printf("Item %d - Weight: ", i + 1);

scanf("%d", &items[i].weight);

items[i].ratio = (float)items[i].value / items[i].weight;

}

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

scanf("%d", &capacity);

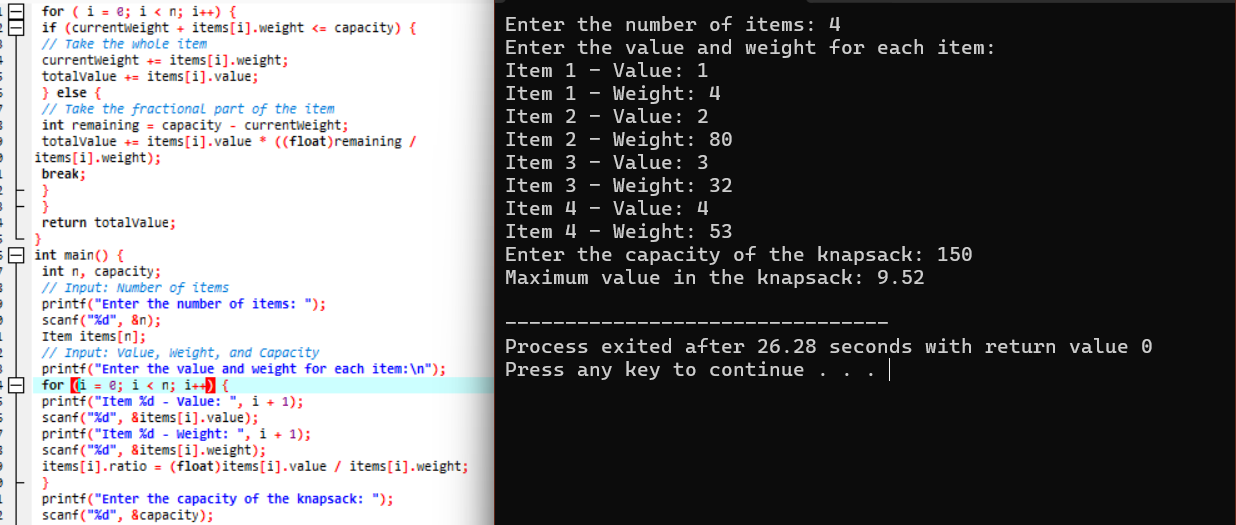
// Compute the maximum value that can be carried

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

printf("Maximum value in the knapsack: %.2f\n", maxValue);

return 0;

}



20.Write a program to perform MST using greedy techniques.

#include <stdio.h>

#include <limits.h>

#define V 5 // Number of vertices in the graph

int v=0,i=0,count=0;

// Function to find the vertex with the minimum key value, from the set of vertices not yet included in the MST

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

int min = INT\_MAX, min\_index;

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

if (mstSet[v] == 0 && key[v] < min)

min = key[v], min\_index = v;

return min\_index;

}

// Function to print the constructed MST stored in parent[]

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

printf("Edge \tWeight\n");

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

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

}

// Function to construct and print MST for a graph represented using an adjacency matrix

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

int parent[V]; // Array to store constructed MST

int key[V]; // Key values used to pick the minimum weight edge in cut

int mstSet[V]; // To represent set of vertices included in MST

// Initialize all keys as INFINITE

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

key[i] = INT\_MAX, mstSet[i] = 0;

// Always include the first vertex in MST.

key[0] = 0; // Make key 0 so that this vertex is picked first

parent[0] = -1; // First node is always the root of MST

// The MST will have V vertices

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

// Pick the minimum key vertex from the set of vertices not yet included in MST

int u = minKey(key, mstSet);

// Add the picked vertex to the MST Set

mstSet[u] = 1;

// Update key value and parent index of the adjacent vertices of the picked vertex

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

// graph[u][v] is non-zero only for adjacent vertices of u

// mstSet[v] is false for vertices not yet included in MST

// Update the key only if graph[u][v] is smaller than key[v]

if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v])

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

}

}

// Print the constructed MST

printMST(parent, graph);

}

int main() {

// Adjacency matrix representation of a graph

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}

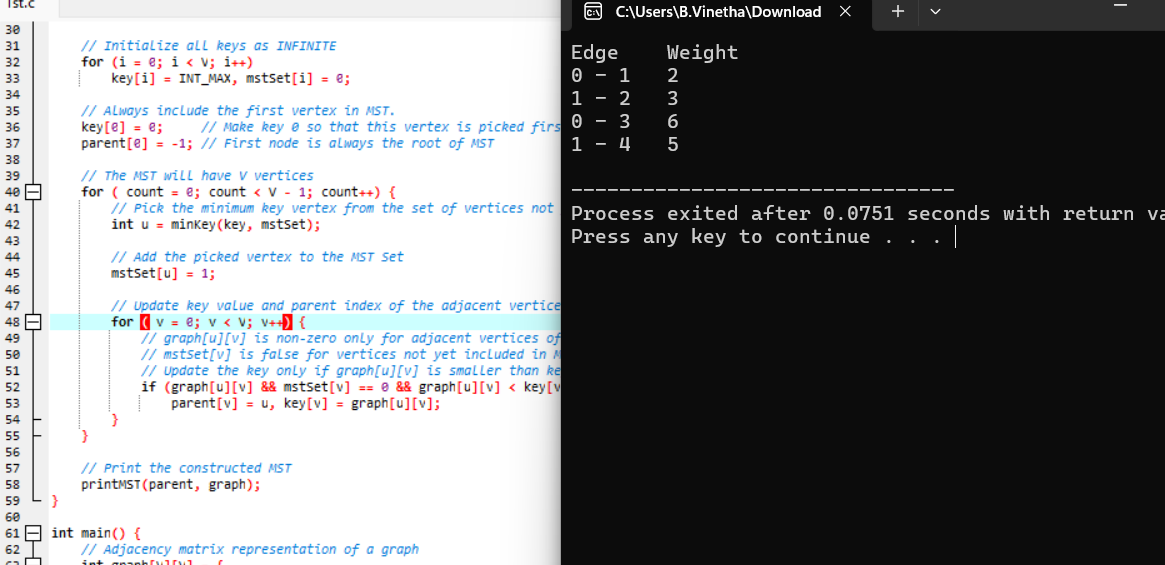
};

// Print the solution

primMST(graph);

return 0;

}



21.Using Dynamic programming concept to find out Optimal binary search tree.

#include <stdio.h>

#include <limits.h>

int i=0,len=0,k=0,j=0;

// Function to find the cost of the optimal binary search tree

void optimalBST(float p[], int n) {

// Tables for storing minimum cost, root and weight

float e[n][n], w[n][n];

int root[n][n];

// Initialize tables

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

e[i][i] = p[i];

w[i][i] = p[i];

root[i][i] = i;

}

// Fill tables for chains of increasing length

for ( len = 2; len <= n; len++) {

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

int j = i + len - 1;

e[i][j] = INT\_MAX;

w[i][j] = w[i][j - 1] + p[j];

// Try different roots and find the minimum cost

for ( k = i; k <= j; k++) {

float t = (k > i ? e[i][k - 1] : 0) + (k < j ? e[k + 1][j]

: 0) + w[i][j];

if (t < e[i][j]) {

e[i][j] = t;

root[i][j] = k;

}

}

}

}

// Print the minimum cost

printf("Minimum cost of the optimal BST: %.2f\n", e[0][n - 1]);

// Print the root table

printf("Root table:\n");

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

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

printf("%2d ", root[i][j]);

}

printf("\n");

}

}

int main() {

int n;

// Input: Number of keys

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

scanf("%d", &n);

float p[n];

// Input: Probabilities of the keys

printf("Enter the probabilities of the keys:\n");

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

printf("Probability of key %d: ", i + 1);

scanf("%f", &p[i]);

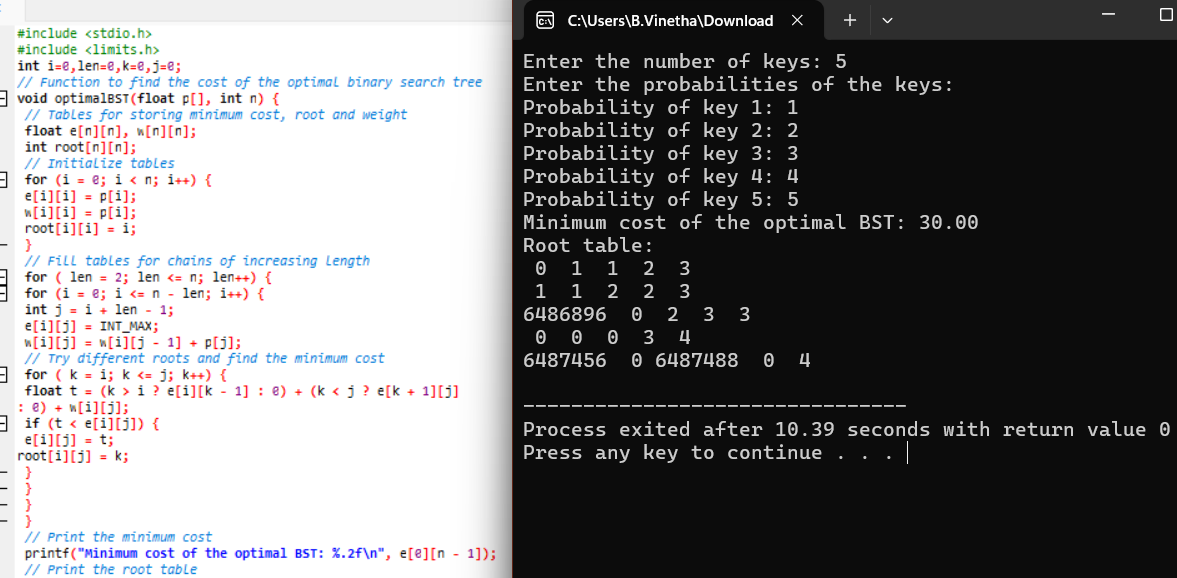
}

// Compute the optimal BST

optimalBST(p, n);

return 0;

}



22.Using Dynamic programming techniques to find binomial coefficient of a given number

#include <stdio.h>

int i=0,j=0;

// Function to compute binomial coefficient C(n, k)

int binomialCoefficient(int n, int k) {

// Create a table to store binomial coefficients

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

// Initialize the table

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

for (j = 0; j <= (i < k ? i : k); j++) {

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

C[i][j] = 1; // Base cases

} else {

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

}

}

}

return C[n][k];

}

int main() {

int n, k;

// Input: values for n and k

printf("Enter the value of n: ");

scanf("%d", &n);

printf("Enter the value of k: ");

scanf("%d", &k);

// Compute and print the binomial coefficient

if (k > n || k < 0) {

printf("Invalid values for n and k.\n");

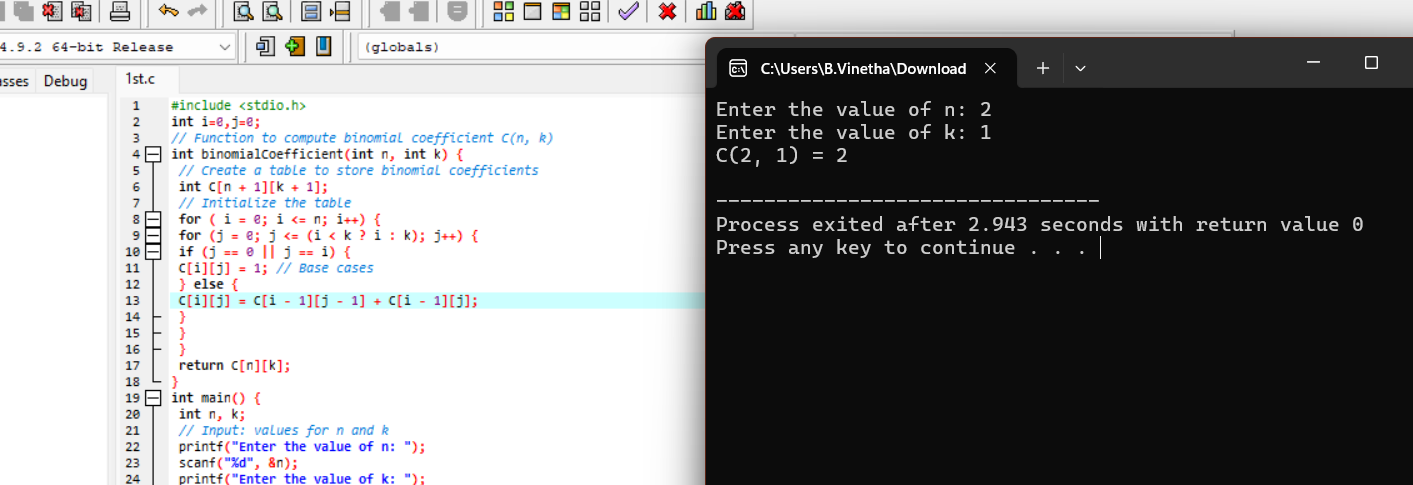
} else {

printf("C(%d, %d) = %d\n", n, k, binomialCoefficient(n, k));

}

return 0;

}



23.Write a program to find the reverse of a given number

#include <stdio.h>

// Function to reverse the digits of a number

int reverseNumber(int num) {

int reversed = 0;

while (num != 0) {

int digit = num % 10; // Extract the last digit

reversed = reversed \* 10 + digit;

num /= 10; // Remove the last digit from the

}

return reversed;

}

int main() {

int number;

// Input: the number to reverse

printf("Enter a number: ");

scanf("%d", &number);

// Reverse the number and print the result

int reversedNumber = reverseNumber(number);

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

return 0;

}



24.Write a program to find the perfect number.

#include <stdio.h>

int i=0;

// Function to check if a number is a perfect number

int isPerfectNumber(int num) {

if (num <= 1) return 0;

int sum = 0;

// Find divisors and sum them up

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

if (num % i == 0) {

sum += i;

}

}

return sum == num;

}

int main() {

int number;

// Input: the number to check

printf("Enter a number: ");

scanf("%d", &number);

// Check if the number is a perfect number and print the result

if (isPerfectNumber(number)) {

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

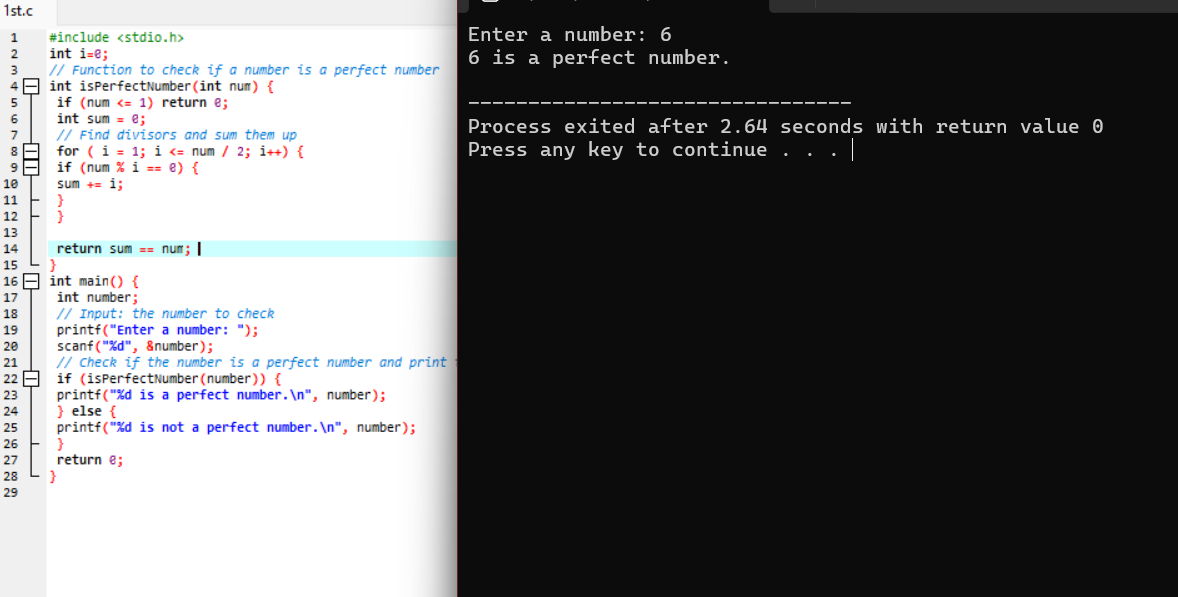
} else {

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

}

return 0;

}



25.Write a program to perform travelling salesman problem using dynamic programming

#include <stdio.h>

#include <limits.h>

#define V 4 // Number of cities

int city=0,i,j;

int tsp(int graph[V][V], int pos, int visited, int dp[V][(1 << V)]) {

// If all cities have been visited, return the cost to go back to the start city

if (visited == (1 << V) - 1)

return graph[pos][0];

// If the subproblem is already solved, return the stored result

if (dp[pos][visited] != -1)

return dp[pos][visited];

int ans = INT\_MAX;

// Try visiting each city that hasn't been visited yet

for ( city = 0; city < V; city++) {

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

// Calculate the cost of visiting the city and recursively solve for the remaining cities

int newAns = graph[pos][city] + tsp(graph, city, visited | (1 << city), dp);

ans = (ans < newAns) ? ans : newAns; // Take the minimum cost

}

}

// Store the result in dp array

return dp[pos][visited] = ans;

}

int main() {

// Adjacency matrix representation of the graph

int graph[V][V] = {

{0, 10, 15, 20},

{10, 0, 35, 25},

{15, 35, 0, 30},

{20, 25, 30, 0}

};

// Create a DP table and initialize it to -1

int dp[V][1 << V];

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

for ( j = 0; j < (1 << V); j++) {

dp[i][j] = -1;

}

}

// Solve the TSP starting from city 0 with the first city visited

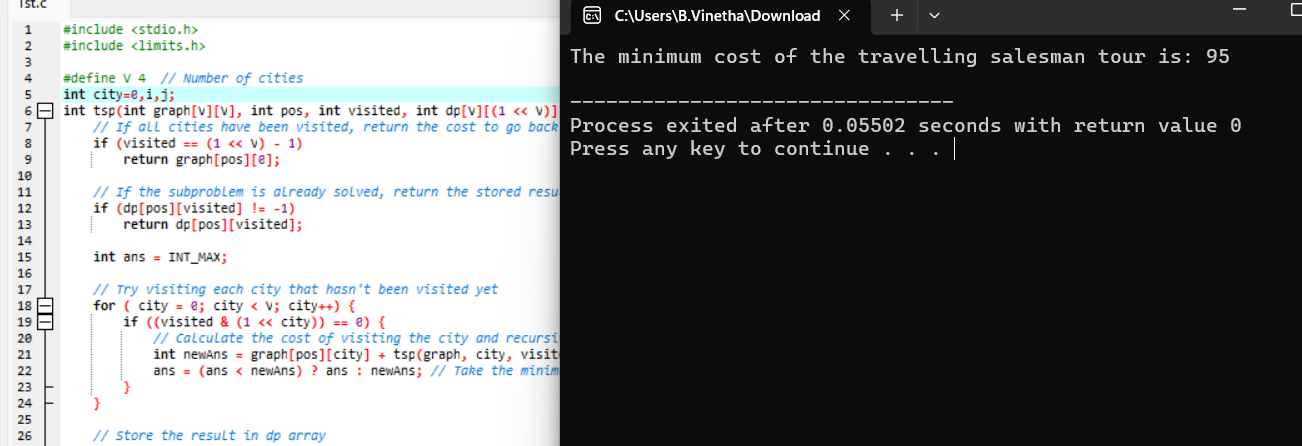
int result = tsp(graph, 0, 1, dp);

// Output the minimum cost

printf("The minimum cost of the travelling salesman tour is: %d\n", result)

return 0;

}



26.Write a program for the given pattern If n=4

#include <stdio.h>

// Function to print the pattern

void printPattern(int n) {

int i,j,k;

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

// Print leading spaces

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

printf(" ");

}

// Print numbers in the current row

for ( k = 1; k <= i; k++) {

printf("%d ", k);

}

printf("\n"); // Move to the next line

}

}

int main() {

int n;

// Input: number of rows

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

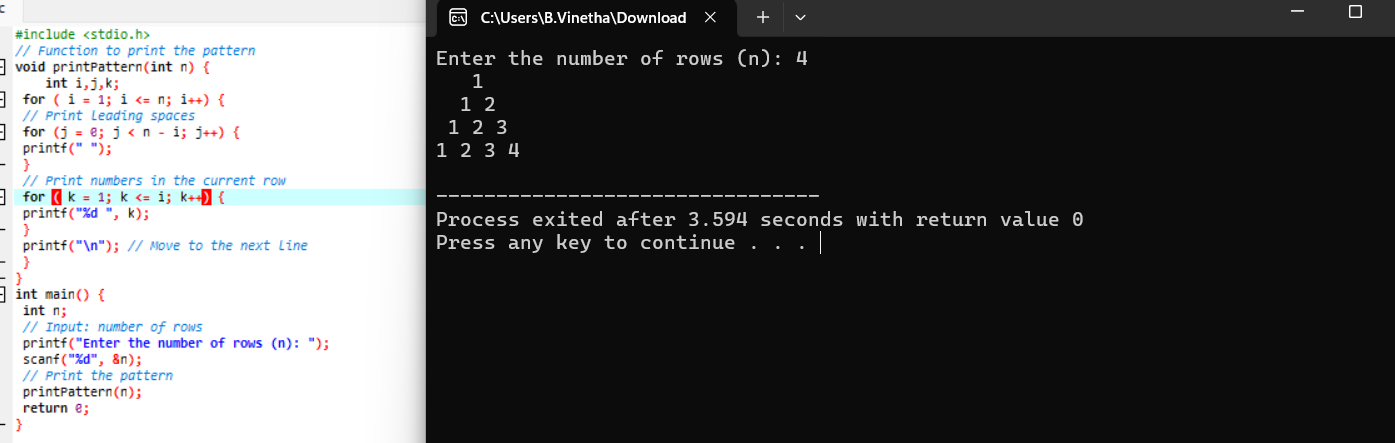
scanf("%d", &n);

// Print the pattern

printPattern(n);

return 0;

}



27.Write a program to perform Floyd’s algorithm

#include <stdio.h>

#include <limits.h>

#define MAX 100

#define INF INT\_MAX

// Function to perform Floyd-Warshall algorithm

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

int dist[MAX][MAX];

int i,j,k;

// Initialize distance matrix

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

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

if (i == j) {

dist[i][j] = 0;

} else if (graph[i][j] != 0) {

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

} else {

dist[i][j] = INF;

}

}

}

// Floyd-Warshall algorithm

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

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

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

if (dist[i][k] != INF && dist[k][j] != INF && dist[i][j] >

dist[i][k] + dist[k][j]) {

dist[i][j] = dist[i][k] + dist[k][j];

}

}

}

}

// Print the distance matrix

printf("Shortest distances between every pair of vertices:\n");

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

for ( 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;

int i,j,k;

// Input: number of vertices

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

scanf("%d", &n);

int graph[MAX][MAX];

// Input: adjacency matrix

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

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

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

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

if (i != j && graph[i][j] == 0) {

graph[i][j] = INF; // Treat zero as infinity for nondiagonal elements

}

}

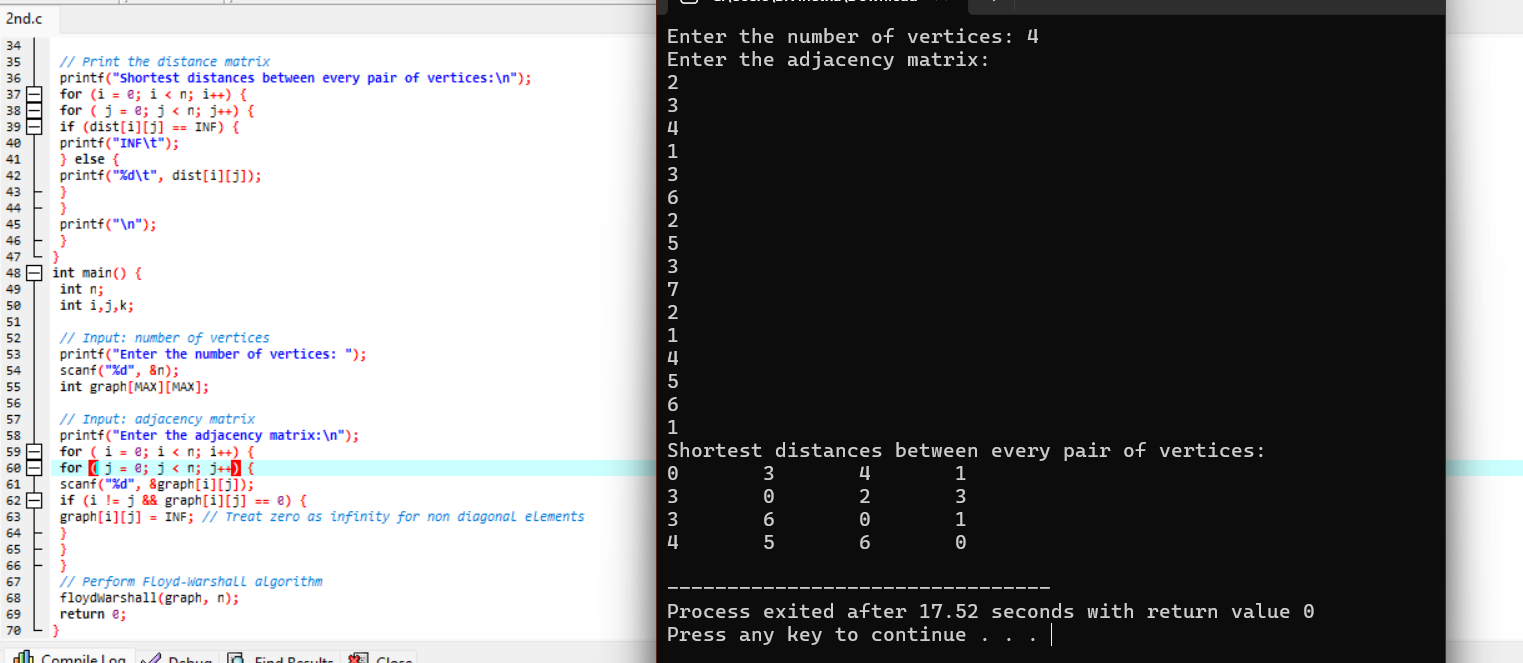
}

// Perform Floyd-Warshall algorithm

floydWarshall(graph, n);

return 0;

}



28.Write a program for pascal triangle.

#include <stdio.h>

// Function to generate Pascal's Triangle

void printPascalsTriangle(int n) {

int triangle[n][n];

int i,j;// Declare a 2D array to store Pascal's Triangle

// Initialize the Pascal's Triangle

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

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

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

triangle[i][j] = 1; // First and last element of each row

} else {

triangle[i][j] = triangle[i - 1][j - 1] + triangle[i -

1][j]; // Sum of two elements above

}

}

}

// Print Pascal's Triangle

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

// Print leading spaces for alignment

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

printf(" ");

}

// Print elements of the current row

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

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

}

printf("\n"); // Move to the next line

}

}

int main() {

int n;

// Input: number of rows

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

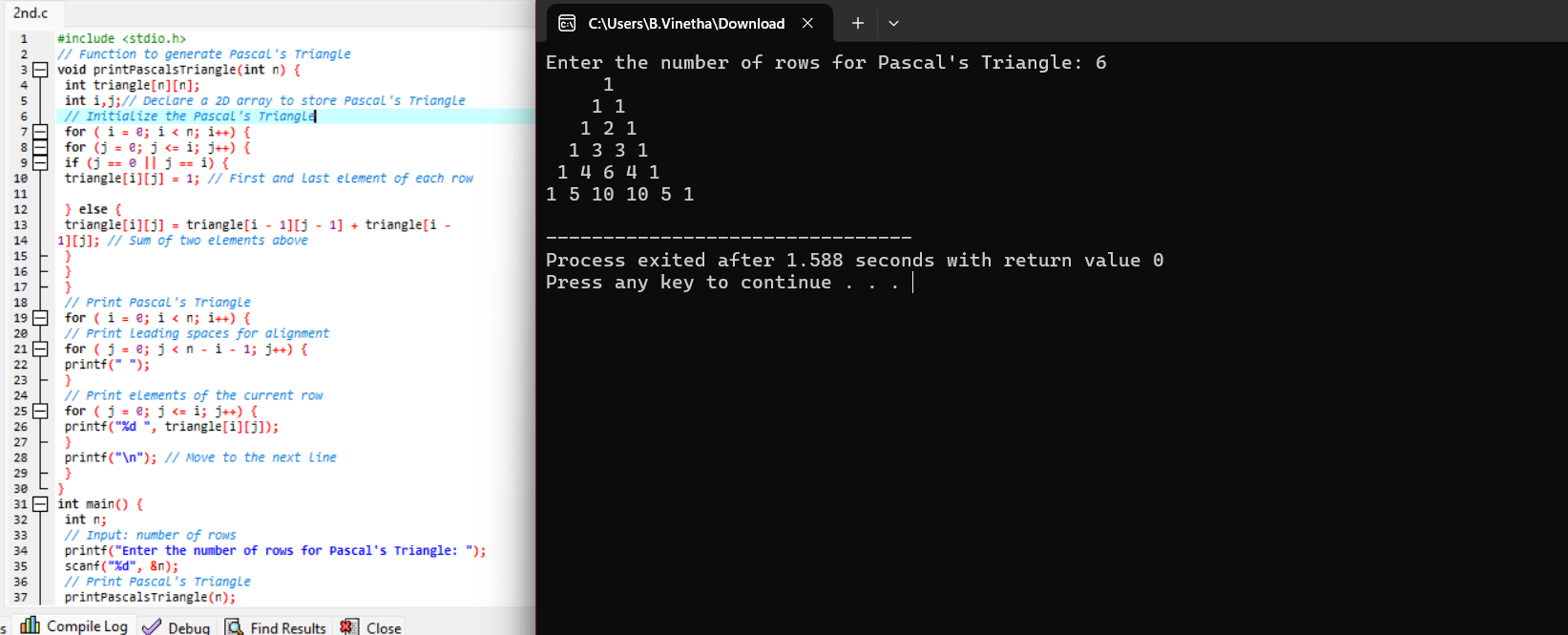
scanf("%d", &n);

// Print Pascal's Triangle

printPascalsTriangle(n);

return 0;

}



29.Write a program to find the optimal cost by using appropriate algorithm

#include <stdio.h>

// Function to find the maximum value in the knapsack

int knapsack(int W, int weights[], int values[], int n) {

int dp[n + 1][W + 1];

int i,w;

// Initialize the dp array

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

for ( w = 0; w <= W; w++) {

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

dp[i][w] = 0;

} else if (weights[i - 1] <= w) {

dp[i][w] = (values[i - 1] + dp[i - 1][w - weights[i - 1]] >

dp[i - 1][w]) ?

(values[i - 1] + dp[i - 1][w - weights[i - 1]])

:

dp[i - 1][w];

} else {

dp[i][w] = dp[i - 1][w];

}

}

}

// The maximum value that can be carried is in dp[n][W]

return dp[n][W];

}

int main() {

int n, W;

int i;

// Input: number of items

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

scanf("%d", &n);

int weights[n], values[n];

// Input: weights and values of items

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

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

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

}

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

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

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

}

// Input: maximum weight capacity

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

scanf("%d", &W);

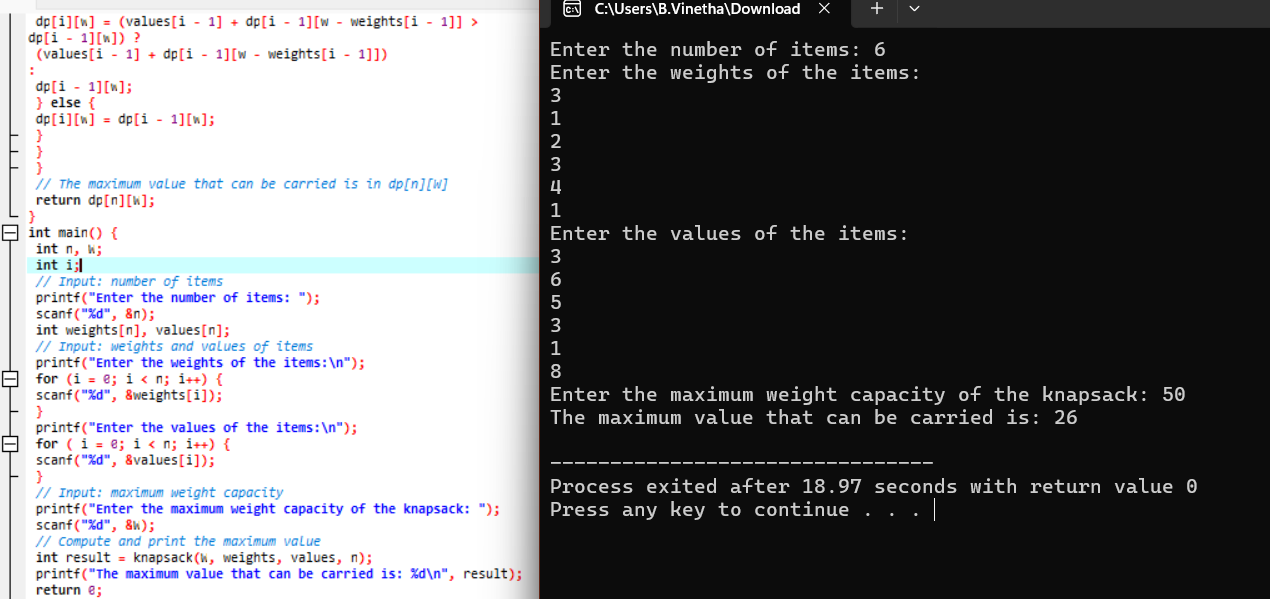
// Compute and print the maximum value

int result = knapsack(W, weights, values, n);

printf("The maximum value that can be carried is: %d\n", result);

return 0;

}



30. Write a program to find the sum of digits

#include <stdio.h>

// Function to calculate the sum of digits of a number

int sumOfDigits(int num) {

int sum = 0;

while (num != 0) {

sum += num % 10; // Add the last digit to sum

num /= 10; // Remove the last digit from the number

}

return sum;

}

int main() {

int number;

// Input: the number to find the sum of digits

printf("Enter a number: ");

scanf("%d", &number);

// Handle negative numbers

if (number < 0) {

number = -number; // Convert to positive if the number is negative

}

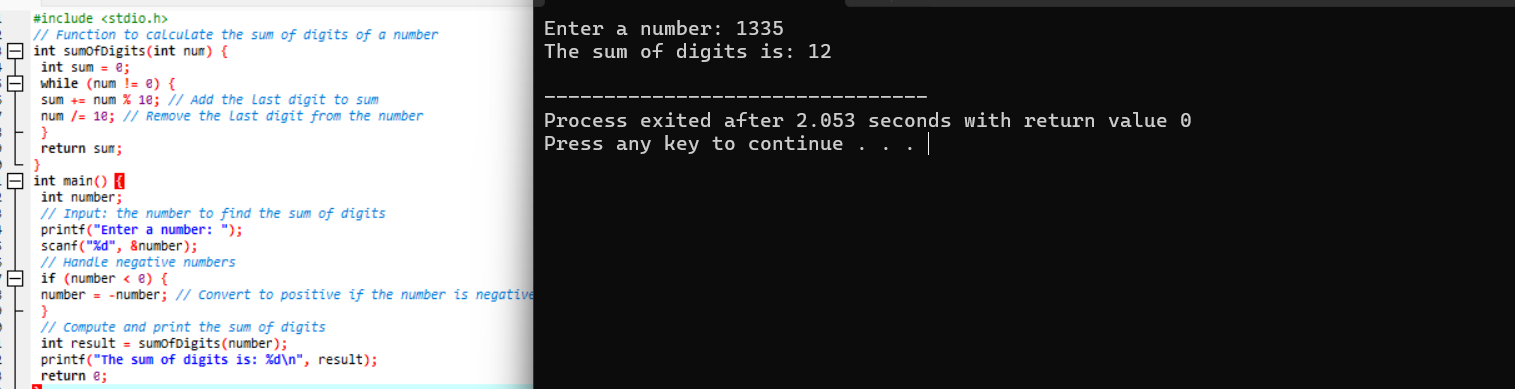
// Compute and print the sum of digits

int result = sumOfDigits(number);

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

return 0;

}



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

#include <stdio.h>

// Function to find the minimum and maximum values in the list

void findMinMax(int arr[], int size, int \*min, int \*max) {

\*min = arr[0];

\*max = arr[0];

int i;

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

if (arr[i] < \*min) {

\*min = arr[i];

}

if (arr[i] > \*max) {

\*max = arr[i];

}

}

}

int main() {

int n;

int i;

// Input: number of elements in the list

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

scanf("%d", &n);

int arr[n];

// Input: elements of the list

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

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

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

}

int min, max;

// Find the minimum and maximum values

findMinMax(arr, n, &min, &max);

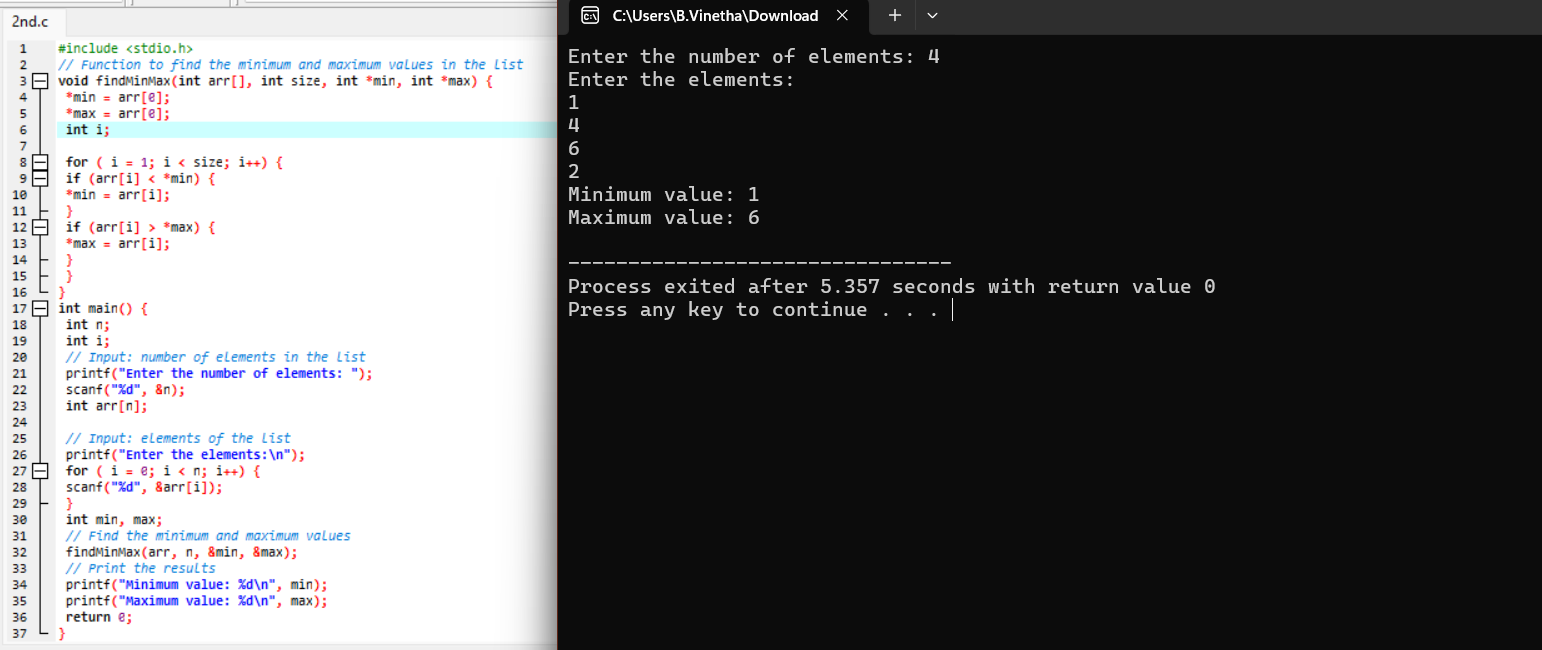
// Print the results

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

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

return 0;

}



32.Write a program to perform n Queens problem using backtracking.

#include <stdio.h>

#include <stdbool.h>

#define MAX 20

// Function to print the solution

void printSolution(int board[MAX][MAX], int N) {

int i,j;

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

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

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

}

printf("\n");

}

printf("\n");

}

// Function to check if a queen can be placed on board[row][col]

bool isSafe(int board[MAX][MAX], int row, int col, int N) {

// Check this column on upper rows

int i,j;

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

if (board[i][col]) {

return false;

}

}

// Check upper left diagonal

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

if (board[i][j]) {

return false;

}

}

// Check upper right diagonal

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

if (board[i][j]) {

return false;

}

}

return true;

}

// Function to solve the N-Queens problem using backtracking

bool solveNQueens(int board[MAX][MAX], int row, int N) {

int col;

if (row >= N) {

return true; // All queens are placed

}

for ( col = 0; col < N; col++) {

if (isSafe(board, row, col, N)) {

board[row][col] = 1; // Place queen

if (solveNQueens(board, row + 1, N)) {

return true; // Recur to place next queen

}

board[row][col] = 0; // Backtrack

}

}

return false; // No solution found

}

int main() {

int N;

int i,j;

int board[MAX][MAX] = {0}; // Initialize board to zero

// Input: number of queens (and size of the board)

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

scanf("%d", &N);

// Solve the N-Queens problem

if (solveNQueens(board, 0, N)) {

printf("One possible solution is:\n");

printSolution(board, N);

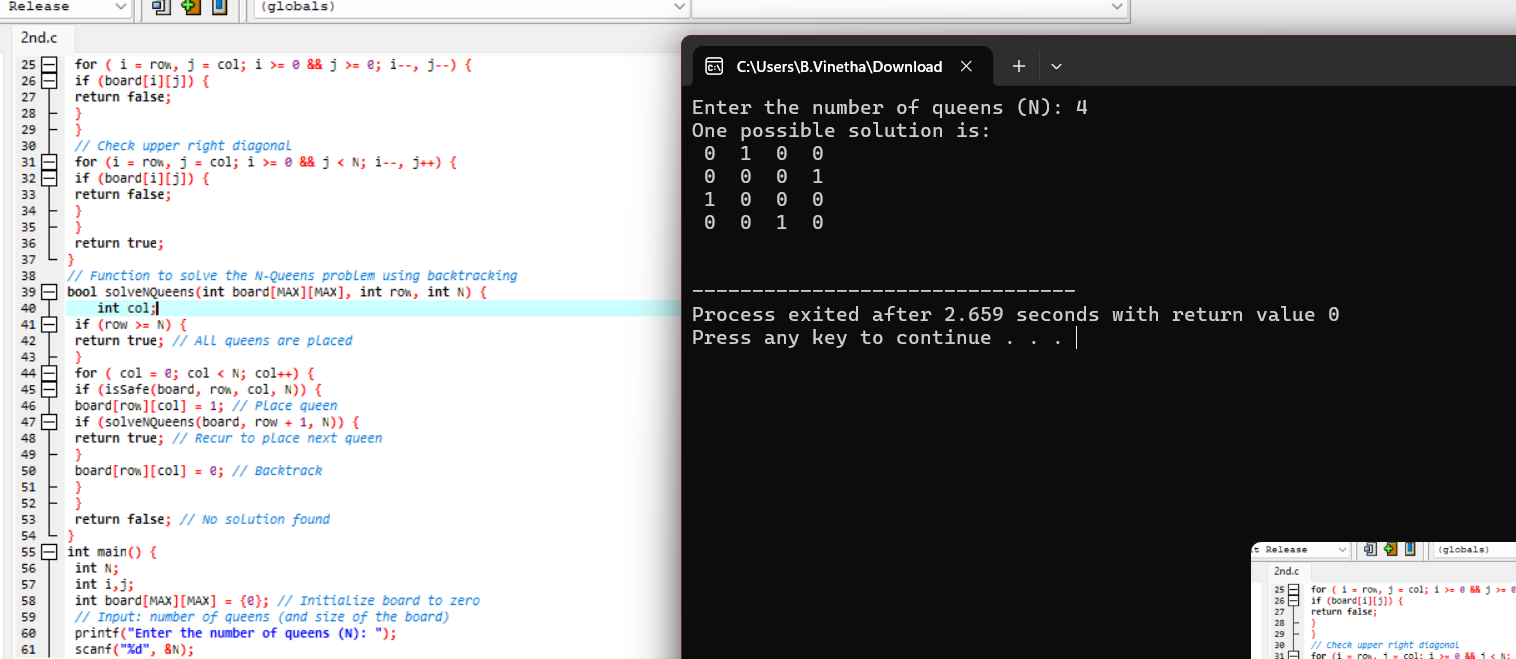
} else {

printf("No solution exists for N = %d\n", N);

}

return 0;

}



33. Write a program to inset a number in a list.

#include <stdio.h>

#define MAX 100 // Define the maximum size of the list

// Function to insert a number into the list at a specific position

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

int i;

// Check if the position is valid

if (position < 0 || position > \*size) {

printf("Invalid position!\n");

return;

}

// Check if the list has enough space

if (\*size >= MAX) {

printf("List is full!\n");

return;

}

// Shift elements to the right to make space for the new number

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

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

}

// Insert the new number at the specified position

list[position] = number;

// Update the size of the list

(\*size)++;

}

// Function to print the list

void printList(int list[], int size) {

int i;

printf("List elements are:\n");

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

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

}

printf("\n");

}

int main() {

int list[MAX];

int i;

int size = 0; // Current number of elements in the list

int number, position;

// Input: number of elements in the list

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

scanf("%d", &size);

// Input: elements of the list

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

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

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

}

// Input: number to insert and the position

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

scanf("%d", &number);

printf("Enter the position to insert the number at (0-based index): ");

scanf("%d", &position);

// Insert the number into the list

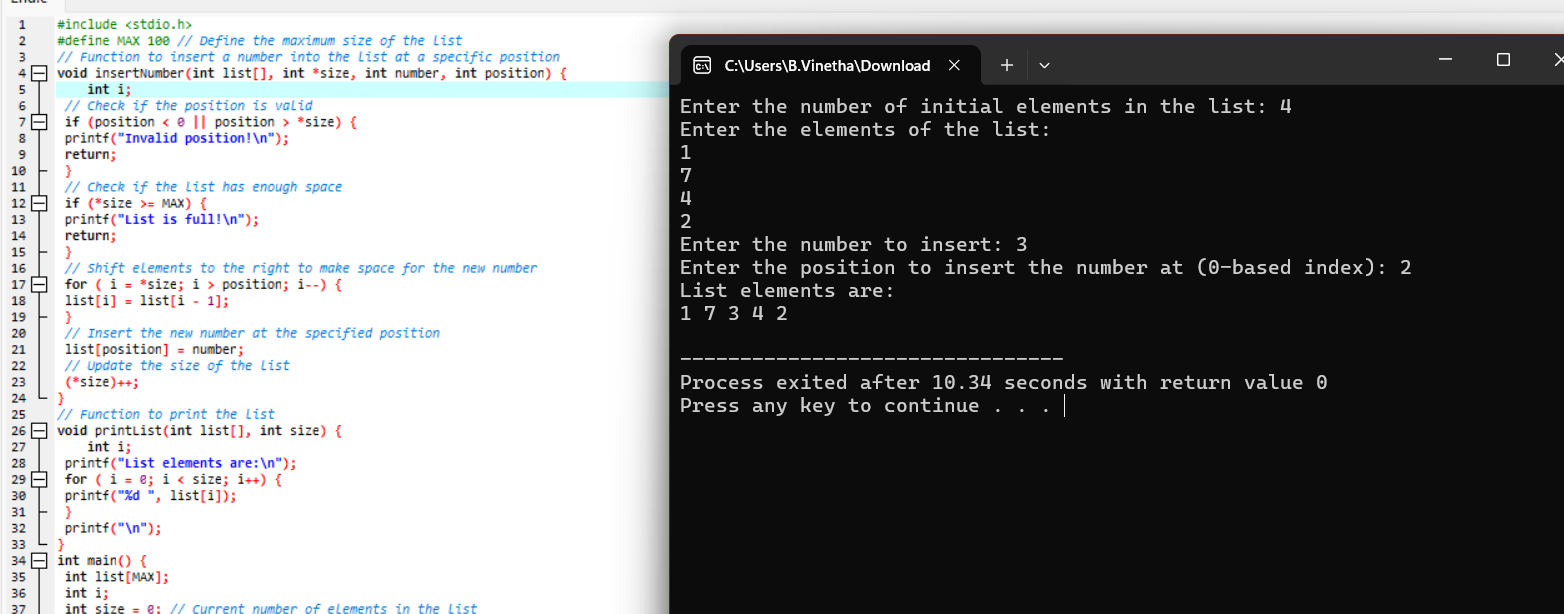
insertNumber(list, &size, number, position);

// Print the updated list

printList(list, size);

return 0;

}



34.Write a program to perform sum of subsets problem using backtracking

#include <stdio.h>

#define MAX 20

// Function to print a subset

void printSubset(int subset[], int size) {

int i;

printf("{ ");

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

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

}

printf("}\n");

}

// Recursive function to find subsets that sum up to the target

void findSubsets(int arr[], int n, int index, int target, int currentSum,

int subset[], int subsetSize) {

// If we have reached the target sum, print the subset

if (currentSum == target) {

printSubset(subset, subsetSize);

return;

}

// If we have processed all elements or exceeded the target sum, return

if (index >= n || currentSum > target) {

return;

}

// Include the current element in the subset

subset[subsetSize] = arr[index];

findSubsets(arr, n, index + 1, target, currentSum + arr[index], subset,

subsetSize + 1);

// Exclude the current element from the subset

findSubsets(arr, n, index + 1, target, currentSum, subset, subsetSize);

}

int main() {

int i;

int arr[MAX], n, target;

int subset[MAX];

// Input: number of elements

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

scanf("%d", &n);

// Input: elements of the set

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

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

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

}

// Input: target sum

printf("Enter the target sum: ");

scanf("%d", &target);

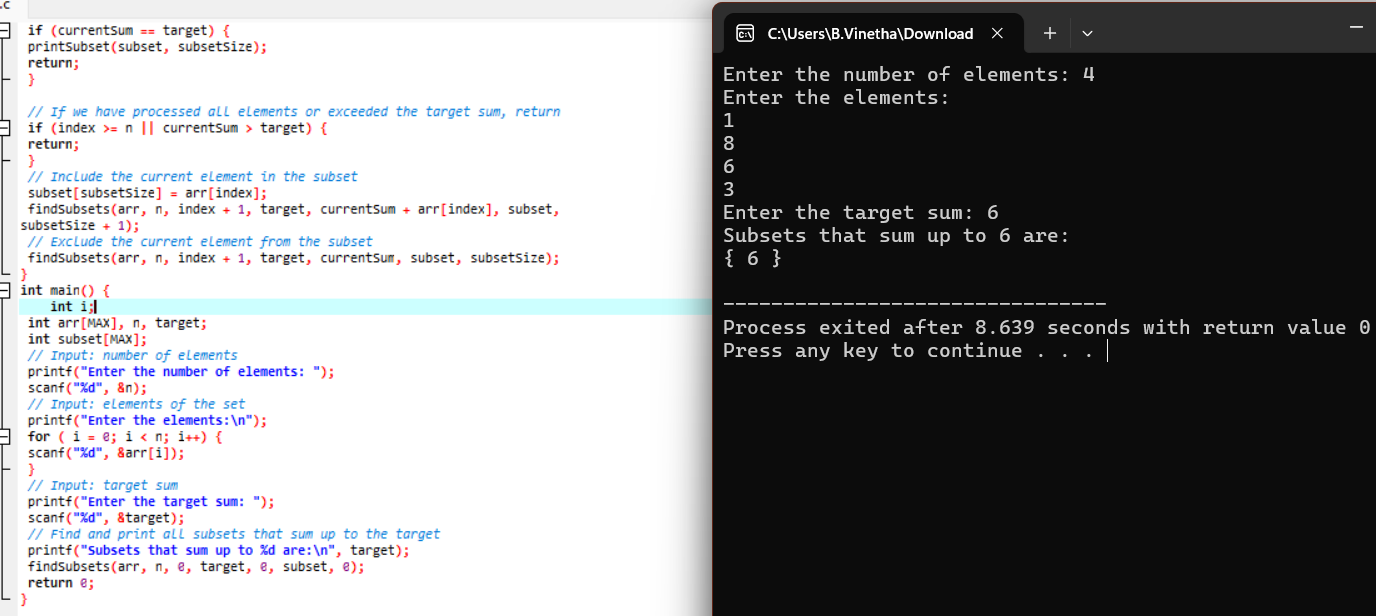
// Find and print all subsets that sum up to the target

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

findSubsets(arr, n, 0, target, 0, subset, 0);

return 0;

}



35.Write a program to perform graph coloring problem using backtracking.

#include <stdio.h>

#include <stdbool.h>

#define MAX\_VERTICES 100

// Function to check if the current color assignment is safe for vertex `v`

bool isSafe(int graph[MAX\_VERTICES][MAX\_VERTICES], int color[], int v, int

c, int V) {

int i,j,k;

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

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

return false;

}

}

return true;

}

// Recursive function to solve the graph coloring problem

bool graphColoringUtil(int graph[MAX\_VERTICES][MAX\_VERTICES], int color[],

int v, int m, int V) {

int c,i,j;

// Base case: If all vertices are assigned a color then return true

if (v == V) {

return true;

}

// Try different colors for vertex `v`

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

if (isSafe(graph, color, v, c, V)) {

color[v] = c; // Assign color `c` to vertex `v`

// Recur to assign colors to the rest of the vertices

if (graphColoringUtil(graph, color, v + 1, m, V)) {

return true;

}

// If assigning color `c` doesn't lead to a solution then

color[v] = 0;

}

}

return false;

}

// Function to solve the graph coloring problem

bool graphColoring(int graph[MAX\_VERTICES][MAX\_VERTICES], int V, int m) {

int i,j,c;

int color[V];

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

color[i] = 0; // Initialize all vertices with no color

}

// Call the recursive function to solve the problem

return graphColoringUtil(graph, color, 0, m, V);

}

// Function to print the color assignment

void printSolution(int color[], int V) {

printf("Solution:\n");

int i,j,c;

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

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

}

}

int main() {

int V, E, m;

int i,color;

int graph[MAX\_VERTICES][MAX\_VERTICES] = {0};

// Input: number of vertices

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

scanf("%d", &V);

// Input: number of edges

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

scanf("%d", &E);

// Input: edges of the graph

printf("Enter the edges (format: u v):\n");

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

int u, v;

scanf("%d %d", &u, &v);

graph[u][v] = 1;

graph[v][u] = 1;

}

// Input: number of colors

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

scanf("%d", &m);

// Solve the graph coloring problem

if (graphColoring(graph, V, m)) {

printf("Solution exists with %d colors:\n", m);

printSolution(color, V);

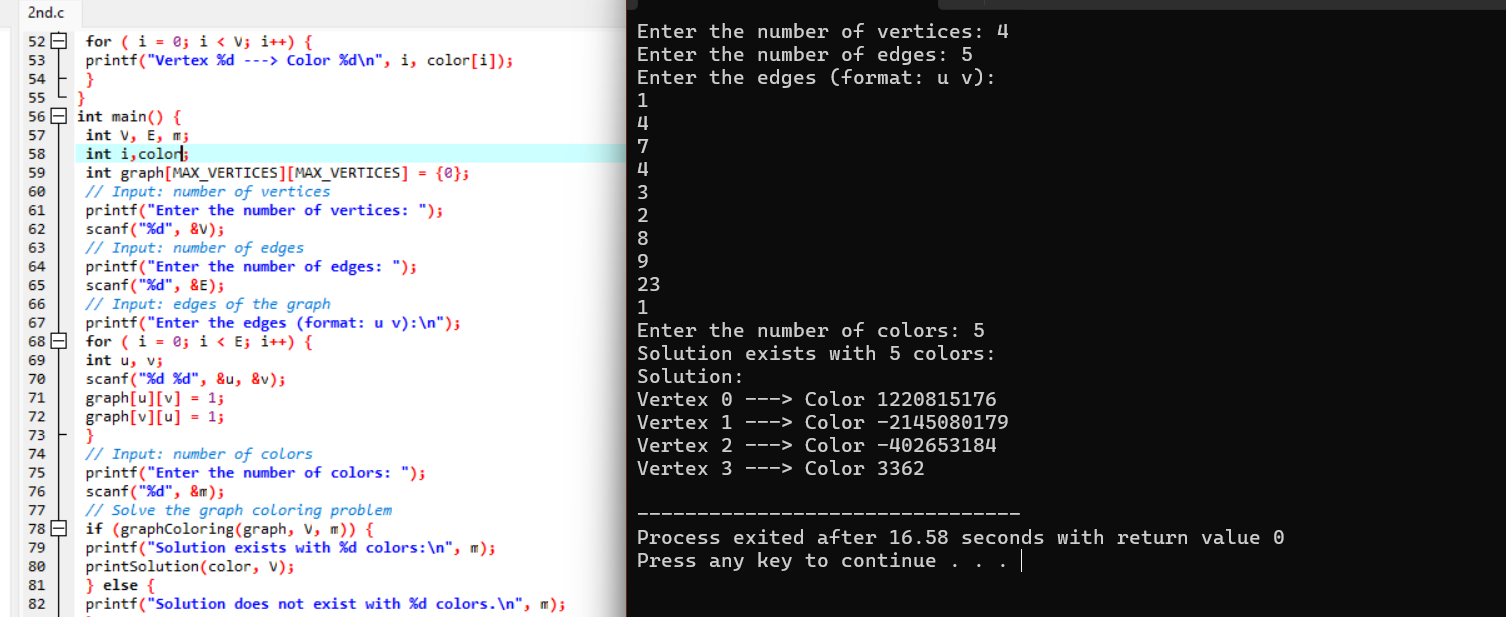
} else {

printf("Solution does not exist with %d colors.\n", m);

}

return 0;

}



36.Write a program to compute container loader Problem.

#include <stdio.h>

#include <stdlib.h>

#define MAX\_ITEMS 100

#define MAX\_BINS 100

// Function to compare items for sorting (used in qsort)

int compare(const void \*a, const void \*b) {

return (\*(int\*)b - \*(int\*)a); // Sort in descending order

}

// Function to perform the First-Fit Decreasing algorithm

void containerLoader(int items[], int n, int binCapacity) {

int bins[MAX\_BINS];

int binCount = 0;

int i, j;

// Initialize all bins to 0

for (i = 0; i < MAX\_BINS; i++) {

bins[i] = 0;

}

// Sort items in decreasing order

qsort(items, n, sizeof(int), compare);

// Place each item into the first bin that can accommodate it

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

int item = items[i];

int placed = 0;

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

if (bins[j] + item <= binCapacity) {

bins[j] += item;

placed = 1;

break;

}

}

// If item was not placed in any bin, create a new bin

if (!placed) {

bins[binCount] = item;

binCount++;

}

}

// Print the results

printf("Number of bins used: %d\n", binCount);

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

printf("Bin %d: %d\n", i + 1, bins[i]);

}

}

int main() {

int items[MAX\_ITEMS];

int i,j;

int n, binCapacity;

// Input: number of items

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

scanf("%d", &n);

// Input: items

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

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

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

}

// Input: bin capacity

printf("Enter the bin capacity: ");

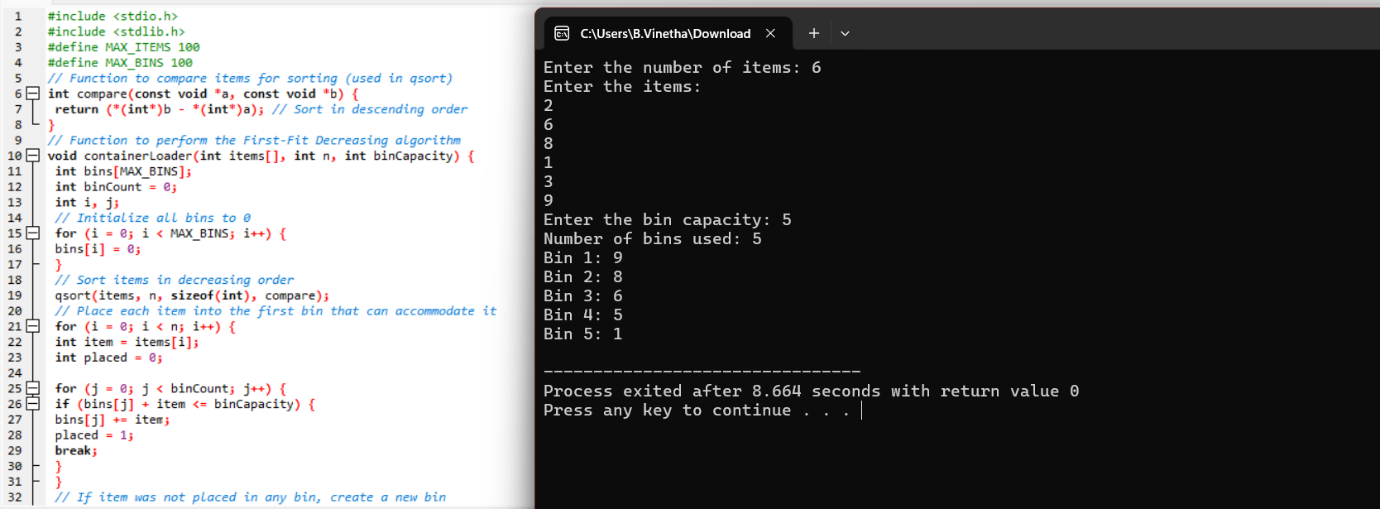
scanf("%d", &binCapacity);

// Solve the container loader problem

containerLoader(items, n, binCapacity);

return 0;

}



37.Write a program to generate the list of all factor for n value.

#include <stdio.h>

// Function to print all factors of a number

void printFactors(int n) {

int i;

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

// Loop through all numbers from 1 to n

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

if (n % i == 0) { // If i is a factor of n

printf("%d ", i);

}

}

printf("\n");

}

int main() {

int n;

// Input: number to find factors for

printf("Enter a number: ");

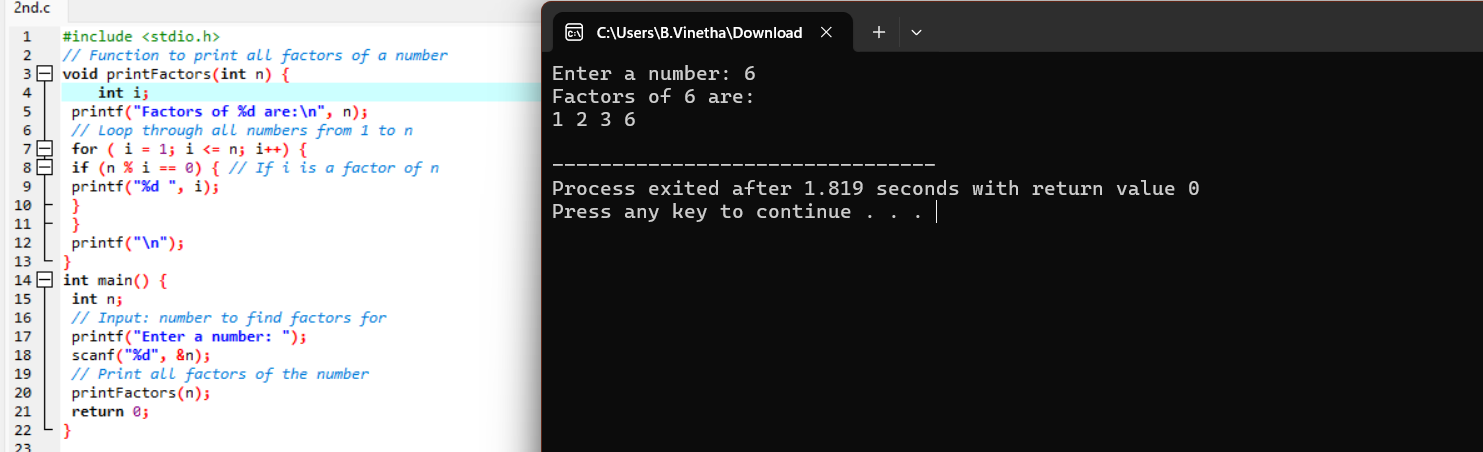
scanf("%d", &n);

// Print all factors of the number

printFactors(n);

return 0;

}



38.Write a program to perform Assignment problem using branch and bound

#include <stdio.h>

#include <limits.h>

#define N 4 // Number of tasks and workers

// Function prototypes

void assignmentProblem(int costMatrix[N][N]);

int branchAndBound(int costMatrix[N][N], int assignment[], int row, int n,

int bound, int currCost, int minCost, int visited[]);

// Helper function to calculate the lower bound for a given node

int calculateLowerBound(int costMatrix[N][N], int assignment[], int n, int

row, int visited[]);

// Helper function to find the minimum cost assignment

int findMinCost(int costMatrix[N][N], int assignment[], int n, int

currCost, int minCost, int visited[]);

int main() {

int costMatrix[N][N] = {

{10, 2, 8, 12},

{9, 4, 7, 6},

{5, 11, 13, 10},

{7, 9, 16, 5}

};

assignmentProblem(costMatrix);

return 0;

}

void assignmentProblem(int costMatrix[N][N]) {

int col,i,j;

int assignment[N] = {-1}; // Store the assignment of tasks to workers

int visited[N] = {0}; // Track visited nodes

int minCost = INT\_MAX; // Initialize minimum cost to a large value

minCost = branchAndBound(costMatrix, assignment, 0, N, 0, 0, minCost,

visited);

printf("Minimum cost is %d\n", minCost);

}

int branchAndBound(int costMatrix[N][N], int assignment[], int row, int n,

int bound, int currCost, int minCost, int visited[]) {

int col;

if (row == n) {

// All tasks are assigned

if (currCost < minCost) {

minCost = currCost;

}

return minCost;

}

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

if (!visited[col]) {

// Mark this worker as visited

visited[col] = 1;

assignment[row] = col;

// Calculate the lower bound for this node

int newBound = bound + costMatrix[row][col];

int lowerBound = calculateLowerBound(costMatrix, assignment, n,

row + 1, visited);

// If the lower bound is less than the minimum cost found so far, explore further

if (newBound + lowerBound < minCost) {

minCost = branchAndBound(costMatrix, assignment, row + 1,

n, newBound, currCost + costMatrix[row][col], minCost, visited);

}

// Backtrack

visited[col] = 0;

assignment[row] = -1;

}

}

return minCost;

}

int calculateLowerBound(int costMatrix[N][N], int assignment[], int n, int

row, int visited[]) {

int i,j;

int bound = 0;

// Calculate the lower bound using row reduction

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

int min1 = INT\_MAX, min2 = INT\_MAX;

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

if (!visited[j] && costMatrix[i][j] < min1) {

min2 = min1;

min1 = costMatrix[i][j];

} else if (!visited[j] && costMatrix[i][j] < min2) {

min2 = costMatrix[i][j];

}

}

bound += (min1 == INT\_MAX) ? 0 : min1;

bound += (min2 == INT\_MAX) ? 0 : min2;

}

// Calculate the lower bound using column reduction

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

int min1 = INT\_MAX, min2 = INT\_MAX;

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

if (!visited[j] && costMatrix[i][j] < min1) {

min2 = min1;

min1 = costMatrix[i][j];

} else if (!visited[j] && costMatrix[i][j] < min2) {

min2 = costMatrix[i][j];

}

}

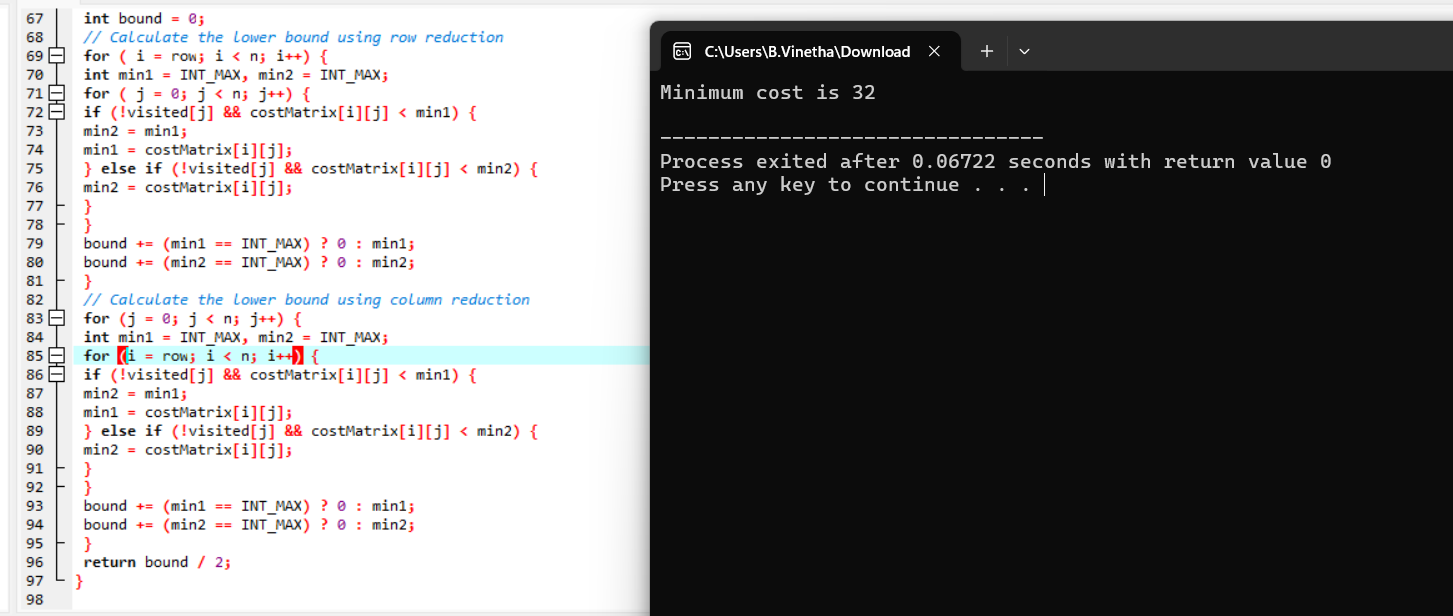
bound += (min1 == INT\_MAX) ? 0 : min1;

bound += (min2 == INT\_MAX) ? 0 : min2;

}

return bound / 2;

}



39.Write a program for to perform liner search.

#include <stdio.h>

// Function to perform linear search

int linearSearch(int arr[], int size, int target) {

int i,j;

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

if (arr[i] == target) {

return i; // Return the index of the target element

}

}

return -1; // Return -1 if the target element is not found

}

int main() {

int arr[100];

int i,j;

int size, target, result;

// Input: size of the array

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

scanf("%d", &size);

// Input: elements of the array

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

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

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

}

// Input: target element to search for

printf("Enter the element to search for: ");

scanf("%d", &target);

// Perform linear search

result = linearSearch(arr, size, target);

// Output the result

if (result != -1) {

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

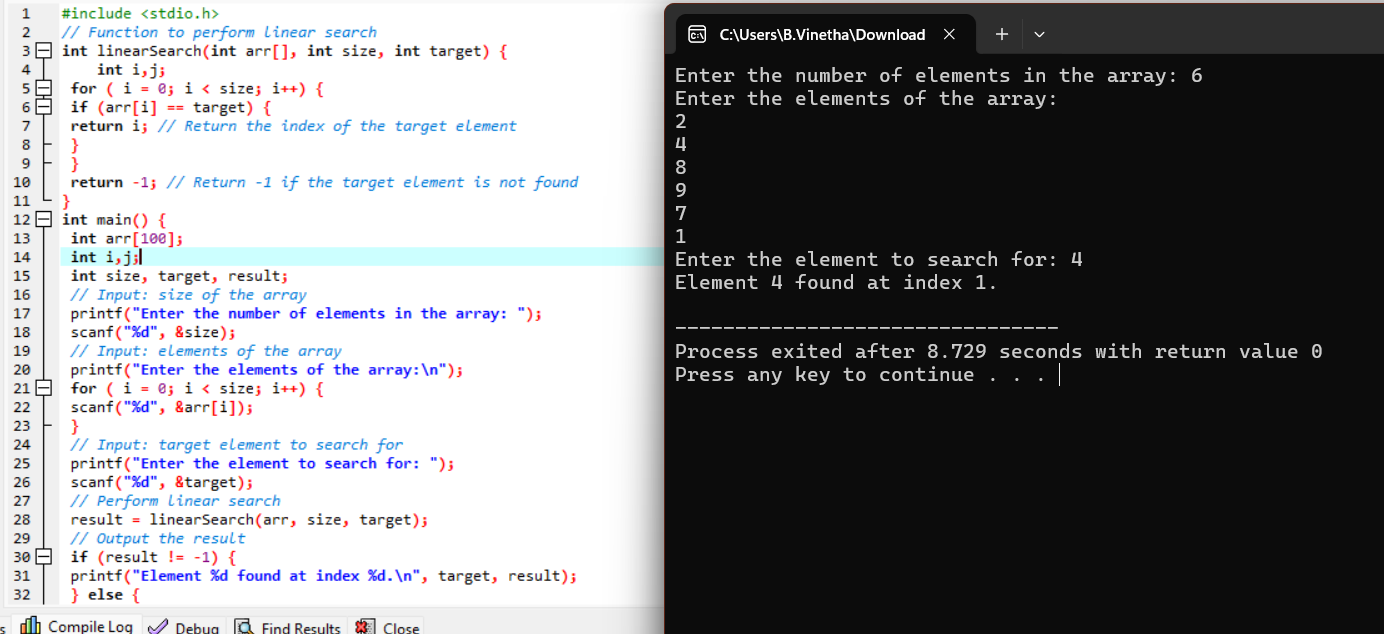
} else {

printf("Element %d not found in the array.\n", target);

}

return 0;

}



40.Write a program to find out Hamiltonian circuit Using backtracking method

#include <stdio.h>

#include <stdbool.h>

#define V 5 // Number of vertices in the graph

// Function to check if the vertex can be added to the Hamiltonian path

bool isSafe(int graph[V][V], int path[], int pos) {

int i,j;

// Check if this vertex is an adjacent vertex of the previously added vertex.

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

return false;

}

// Check if the vertex has already been included.

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

if (path[i] == path[pos]) {

return false;

}

}

return true;

}

// Function to solve the Hamiltonian Circuit problem

bool hamCycleUtil(int graph[V][V], int path[], int pos)

{

int i,j,v;

// Base case: If all vertices are included in the path

if (pos == V) {

// And if there is an edge from the last included vertex to the first vertex

return graph[path[pos-1]][path[0]] == 1;

}

// Try different vertices as the next candidate in the path

for ( v = 1; v < V; v++) {

if (isSafe(graph, path, pos)) {

path[pos] = v;

// Recur to build the rest of the path

if (hamCycleUtil(graph, path, pos + 1)) {

return true;

}

// If adding vertex v doesn't lead to a solution, remove it

path[pos] = -1;

}

}

return false;

}

// Function to find and print the Hamiltonian Circuit

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

int path[V];

int i,j,v;

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

path[i] = -1;

}

// Let the first vertex in the path be 0

path[0] = 0;

if (hamCycleUtil(graph, path, 1) == false) {

printf("No Hamiltonian Circuit found\n");

} else {

printf("Hamiltonian Circuit found:\n");

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

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

}

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

}

}

int main() {

// Graph representation using adjacency matrix

int graph[V][V] = {

{0, 1, 1, 1, 0},

{1, 0, 1, 1, 1},

{1, 1, 0, 1, 1},

{1, 1, 1, 0, 1},

{0, 1, 1, 1, 0}

};

findHamiltonianCircuit(graph);

return 0;

}

