1. #include <stdio.h>

int main() {

int operand1, operand2;

// Input operands

printf("Enter operand 1: ");

scanf("%d", &operand1);

printf("Enter operand 2: ");

scanf("%d", &operand2);

// Arithmetic operators

printf("Addition (+): %d\n", operand1 + operand2);

printf("Subtraction (-): %d\n", operand1 - operand2);

printf("Multiplication (\*): %d\n", operand1 \* operand2);

printf("Division (/): %d\n", operand1 / operand2);

printf("Modulus (%%): %d\n", operand1 % operand2);

// Increment and Decrement operators

printf("Pre-increment (++operand1): %d\n", ++operand1);

printf("Post-increment (operand1++): %d\n", operand1++);

printf("Pre-decrement (--operand2): %d\n", --operand2);

printf("Post-decrement (operand2--): %d\n", operand2--);

// Relational operators

printf("Equal to (==): %d\n", operand1 == operand2);

printf("Not equal to (!=): %d\n", operand1 != operand2);

printf("Greater than (>): %d\n", operand1 > operand2);

printf("Less than (<): %d\n", operand1 < operand2);

printf("Greater than or equal to (>=): %d\n", operand1 >= operand2);

printf("Less than or equal to (<=): %d\n", operand1 <= operand2);

// Logical operators

printf("Logical AND (&&): %d\n", operand1 && operand2);

printf("Logical OR (||): %d\n", operand1 || operand2);

printf("Logical NOT (!operand1): %d\n", !operand1);

// Bitwise operators

printf("Bitwise AND (&): %d\n", operand1 & operand2);

printf("Bitwise OR (|): %d\n", operand1 | operand2);

printf("Bitwise XOR (^): %d\n", operand1 ^ operand2);

printf("Bitwise NOT (~operand1): %d\n", ~operand1);

return 0;

}

2.#include <stdio.h>

int main() {

float floatNumber;

// Input a floating-point number

printf("Enter a floating-point number: ");

scanf("%f", &floatNumber);

// Automatic type conversion (float to int)

int intResultAuto = floatNumber;

// Type casting (float to int)

int intResultCast = (int)floatNumber;

// Display results

printf("\nResults:\n");

printf("Original floating-point number: %.2f\n", floatNumber);

printf("After auto conversion (float to int): %d\n", intResultAuto);

printf("After casting (float to int): %d\n", intResultCast);

return 0;

}

1. #include <stdio.h>

int main() {

int num1, num2, num3;

// Input three numbers

printf("Enter the first number: ");

scanf("%d", &num1);

printf("Enter the second number: ");

scanf("%d", &num2);

printf("Enter the third number: ");

scanf("%d", &num3);

// Finding the maximum and minimum

int max, min;

// Assume the first number is both max and min initially

max = min = num1;

// Check and update for the second number

if (num2 > max) {

max = num2;

} else if (num2 < min) {

min = num2;

}

// Check and update for the third number

if (num3 > max) {

max = num3;

} else if (num3 < min) {

min = num3;

}

// Display results

printf("\nResults:\n");

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

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

return 0;

}

1. #include <stdio.h>

#include <math.h>

int main() {

// Variables for principal, rate, time, and interest

float principal, rate, time;

float simpleInterest, compoundInterest;

// Input principal amount

printf("Enter the principal amount: ");

scanf("%f", &principal);

// Input annual interest rate

printf("Enter the annual interest rate (in percentage): ");

scanf("%f", &rate);

// Input time in years

printf("Enter the time (in years): ");

scanf("%f", &time);

// Simple Interest calculation

simpleInterest = (principal \* rate \* time) / 100;

// Compound Interest calculation

compoundInterest = principal \* (pow(1 + rate / 100, time) - 1);

// Display results

printf("\nResults:\n");

printf("Principal amount: %.2f\n", principal);

printf("Annual interest rate: %.2f%%\n", rate);

printf("Time (in years): %.2f\n", time);

printf("Simple Interest: %.2f\n", simpleInterest);

printf("Compound Interest: %.2f\n", compoundInterest);

return 0;

}

1. #include <stdio.h>

int main() {

float percentage;

// Input percentage of marks

printf("Enter the percentage of marks: ");

scanf("%f", &percentage);

// Class awarded based on percentage

if (percentage < 40) {

printf("Class: Failed\n");

} else if (percentage > 60) {

printf("Class: Second class\n");

} else if (percentage > 70) {

printf("Class: First class\n");

} else {

printf("Class: Distinction\n");

}

return 0;

}

1. #include <stdio.h>

void multiplication\_table(int number, int rows) {

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

int result = number \* i;

printf("%d x %d = %d\n", number, i, result);

}

}

int main() {

int number, rows;

// Input

printf("Enter the number: ");

scanf("%d", &number);

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

scanf("%d", &rows);

// Print multiplication table

multiplication\_table(number, rows);

return 0;

}

1. #include <stdio.h>

void decimalToBinary(int decimal) {

// Binary representation array

int binary[8];

// Initialize binary array elements to 0

for (int i = 0; i < 8; ++i) {

binary[i] = 0;

}

// Convert decimal to binary

int index = 7; // Start from the rightmost bit

while (decimal > 0 && index >= 0) {

binary[index] = decimal % 2;

decimal /= 2;

--index;

}

// Print binary representation

printf("Binary equivalent: ");

for (int i = 0; i < 8; ++i) {

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

}

printf("\n");

}

int main() {

int decimal;

// Input

printf("Enter a positive number between 0 and 255: ");

scanf("%d", &decimal);

// Check if the input is within the valid range

if (decimal < 0 || decimal > 255) {

printf("Invalid input. Please enter a positive number between 0 and 255.\n");

return 1; // Exit with an error code

}

// Convert and print the binary representation

decimalToBinary(decimal);

return 0;

}

1. #include <stdio.h>

#include <math.h>

#define GRAVITY -9.8

void timeToReachEachFloor(int numberOfFloors, double floorHeight) {

// Calculate time for each floor

for (int floor = 1; floor <= numberOfFloors; ++floor) {

double distance = floor \* floorHeight;

// Using the formula: s = ut + (1/2)at^2

// Since u = 0, the formula simplifies to: s = (1/2)at^2

double time = sqrt((2 \* distance) / fabs(GRAVITY));

printf("Time to reach floor %d: %.2f seconds\n", floor, time);

}

}

int main() {

int numberOfFloors = 10;

double floorHeight = 3.0;

// Calculate and print the time to reach each floor

timeToReachEachFloor(numberOfFloors, floorHeight);

return 0;

}

1. #include <stdio.h>

int main() {

int operand1, operand2, result;

char operator;

// Input

printf("Enter the first operand: ");

scanf("%d", &operand1);

printf("Enter the second operand: ");

scanf("%d", &operand2);

printf("Enter the operator (+, -, \*, /, %%): ");

scanf(" %c", &operator); // Note the space before %c to consume the newline character

// Perform operation based on the operator

switch (operator) {

case '+':

result = operand1 + operand2;

break;

case '-':

result = operand1 - operand2;

break;

case '\*':

result = operand1 \* operand2;

break;

case '/':

// Check for division by zero

if (operand2 != 0) {

result = operand1 / operand2;

} else {

printf("Error: Division by zero is undefined.\n");

return 1; // Exit with an error code

}

break;

case '%':

// Check for modulus by zero

if (operand2 != 0) {

result = operand1 % operand2;

} else {

printf("Error: Modulus by zero is undefined.\n");

return 1; // Exit with an error code

}

break;

default:

printf("Error: Invalid operator.\n");

return 1; // Exit with an error code

}

// Output the result

printf("Result: %d %c %d = %d\n", operand1, operator, operand2, result);

return 0;

}

1. #include <stdio.h>

#include <stdbool.h>

bool isPrime(int number) {

// 0 and 1 are not prime numbers

if (number <= 1) {

return false;

}

// Check for factors from 2 to the square root of the number

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

if (number % i == 0) {

// If the number is divisible by any value in this range, it's not prime

return false;

}

}

// If no factors were found, the number is prime

return true;

}

int main() {

int num;

// Input

printf("Enter a number: ");

scanf("%d", &num);

// Check and output whether the number is prime

if (isPrime(num)) {

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

} else {

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

}

return 0;

}

1. #include <stdio.h>

void generateFibonacci(int n) {

int firstTerm = 0, secondTerm = 1, nextTerm;

printf("Fibonacci Sequence for the first %d terms:\n", n);

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

printf("%d, ", firstTerm);

nextTerm = firstTerm + secondTerm;

firstTerm = secondTerm;

secondTerm = nextTerm;

}

printf("\n");

}

int main() {

int n;

// Input

printf("Enter the number of terms in the Fibonacci sequence: ");

scanf("%d", &n);

// Check for valid input

if (n <= 0) {

printf("Please enter a positive integer for the number of terms.\n");

return 1; // Exit with an error code

}

// Generate and print the Fibonacci sequence

generateFibonacci(n);

return 0;

}

1. #include <stdio.h>

#include <stdbool.h>

// Function to check if a number is prime

bool isPrime(int number) {

if (number <= 1) {

return false;

}

// Check for factors from 2 to the square root of the number

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

if (number % i == 0) {

return false; // If the number is divisible by any value in this range, it's not prime

}

}

return true;

}

// Function to generate and print prime numbers up to n

void generatePrimes(int n) {

printf("Prime numbers between 1 and %d are:\n", n);

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

if (isPrime(i)) {

printf("%d, ", i);

}

}

printf("\n");

}

int main() {

int n;

// Input

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

scanf("%d", &n);

// Check for valid input

if (n < 1) {

printf("Please enter a positive integer for n.\n");

return 1; // Exit with an error code

}

// Generate and print prime numbers up to n

generatePrimes(n);

return 0;

}

1. #include <stdio.h>

#include <math.h>

void findRoots(float a, float b, float c) {

float discriminant, root1, root2;

// Calculate discriminant

discriminant = b \* b - 4 \* a \* c;

// Check if the discriminant is non-negative

if (discriminant >= 0) {

// Calculate roots

root1 = (-b + sqrt(discriminant)) / (2 \* a);

root2 = (-b - sqrt(discriminant)) / (2 \* a);

// Output roots

printf("Root 1: %.2f\n", root1);

printf("Root 2: %.2f\n", root2);

} else {

// If discriminant is negative, roots are complex numbers

float realPart = -b / (2 \* a);

float imaginaryPart = sqrt(fabs(discriminant)) / (2 \* a);

// Output complex roots

printf("Root 1: %.2f + %.2fi\n", realPart, imaginaryPart);

printf("Root 2: %.2f - %.2fi\n", realPart, imaginaryPart);

}

}

int main() {

float a, b, c;

// Input coefficients

printf("Enter the coefficients of the quadratic equation (ax^2 + bx + c = 0):\n");

printf("Enter a: ");

scanf("%f", &a);

printf("Enter b: ");

scanf("%f", &b);

printf("Enter c: ");

scanf("%f", &c);

// Check if 'a' is not equal to zero

if (a == 0) {

printf("Error: 'a' should not be zero for a quadratic equation.\n");

return 1; // Exit with an error code

}

// Find and print the roots

findRoots(a, b, c);

return 0;

}

1. #include <stdio.h>

#include <math.h>

double calculateSeries(double x) {

// Calculate the expression: 1 - x/2 + x^2/4 - x^3/6

double result = 1.0 - x / 2.0 + pow(x, 2) / 4.0 - pow(x, 3) / 6.0;

return result;

}

int main() {

double x;

// Input

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

scanf("%lf", &x);

// Calculate and print the result

double seriesResult = calculateSeries(x);

printf("Result of the series for x = %.2f: %.6f\n", x, seriesResult);

return 0;

}

1. #include <stdio.h>

#include <math.h>

double computeGeometricProgression(double x, int n) {

double sum = 0;

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

sum += pow(x, i);

}

return sum;

}

int main() {

double x;

int n;

// Input

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

scanf("%lf", &x);

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

scanf("%d", &n);

// Check for valid input

if (n < 0) {

printf("Error: 'n' should be a non-negative integer.\n");

return 1; // Exit with an error code

}

// Calculate and print the result

double result = computeGeometricProgression(x, n);

printf("Sum of the geometric progression: %.2f\n", result);

return 0;

}

1. #include <stdio.h>

void findMinMaxAvg(int arr[], int size, int \*min, int \*max, double \*avg) {

// Initialize min and max with the first element of the array

\*min = \*max = arr[0];

\*avg = 0;

// Calculate min, max, and sum

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

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

\*min = arr[i];

}

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

\*max = arr[i];

}

\*avg += arr[i];

}

// Calculate average

\*avg /= size;

}

int main() {

int size;

// Input the size of the array

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

scanf("%d", &size);

// Check for valid input

if (size <= 0) {

printf("Error: Array size should be a positive integer.\n");

return 1; // Exit with an error code

}

int arr[size];

// Input elements of the array

printf("Enter %d integers for the array:\n", size);

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

printf("Enter element %d: ", i + 1);

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

}

int min, max;

double avg;

// Find and print min, max, and average

findMinMaxAvg(arr, size, &min, &max, &avg);

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

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

printf("Average: %.2f\n", avg);

return 0;

}

1. #include <stdio.h>

#include <math.h>

void computeStatistics(int arr[], int n, double \*mean, double \*variance, double \*stdDeviation) {

// Calculate mean

\*mean = 0;

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

\*mean += arr[i];

}

\*mean /= n;

// Calculate variance

\*variance = 0;

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

\*variance += pow(arr[i] - \*mean, 2);

}

\*variance /= n;

// Calculate standard deviation

\*stdDeviation = sqrt(\*variance);

}

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

// Bubble sort

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

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

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

// Swap elements if they are in the wrong order

int temp = arr[j];

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

arr[j + 1] = temp;

}

}

}

}

int main() {

int n;

// Input

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

scanf("%d", &n);

if (n <= 0) {

printf("Error: Number of elements should be a positive integer.\n");

return 1; // Exit with an error code

}

int arr[n];

// Input elements of the array

printf("Enter %d integers for the array:\n", n);

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

printf("Enter element %d: ", i + 1);

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

}

double mean, variance, stdDeviation;

// Compute statistics and sort the array

computeStatistics(arr, n, &mean, &variance, &stdDeviation);

sortArray(arr, n);

// Print the results

printf("Mean: %.2f\n", mean);

printf("Variance: %.2f\n", variance);

printf("Standard Deviation: %.2f\n", stdDeviation);

printf("Sorted Array: ");

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

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

}

printf("\n");

return 0;

}

1. #include <stdio.h>

#include <stdlib.h>

// Function to input a matrix

void inputMatrix(int \*\*matrix, int rows, int cols) {

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

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

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

printf("Enter element at position (%d, %d): ", i + 1, j + 1);

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

}

}

}

// Function to display a matrix

void displayMatrix(int \*\*matrix, int rows, int cols) {

printf("Matrix:\n");

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

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

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

}

printf("\n");

}

}

// Function to add two matrices

void addMatrices(int \*\*matrix1, int \*\*matrix2, int \*\*result, int rows, int cols) {

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

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

result[i][j] = matrix1[i][j] + matrix2[i][j];

}

}

}

// Function to multiply two matrices

void multiplyMatrices(int \*\*matrix1, int rows1, int cols1, int \*\*matrix2, int cols2, int \*\*result) {

for (int i = 0; i < rows1; ++i) {

for (int j = 0; j < cols2; ++j) {

result[i][j] = 0;

for (int k = 0; k < cols1; ++k) {

result[i][j] += matrix1[i][k] \* matrix2[k][j];

}

}

}

}

// Function to transpose a matrix

void transposeMatrix(int \*\*matrix, int rows, int cols, int \*\*result) {

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

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

result[j][i] = matrix[i][j];

}

}

}

int main() {

int rows1, cols1, rows2, cols2;

// Input dimensions of the matrices

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

scanf("%d %d", &rows1, &cols1);

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

scanf("%d %d", &rows2, &cols2);

// Check for valid input

if (cols1 != rows2) {

printf("Error: The number of columns in the first matrix must be equal to the number of rows in the second matrix for multiplication.\n");

return 1; // Exit with an error code

}

// Dynamically allocate memory for matrices

int \*\*matrix1 = (int \*\*)malloc(rows1 \* sizeof(int \*));

for (int i = 0; i < rows1; ++i) {

matrix1[i] = (int \*)malloc(cols1 \* sizeof(int));

}

int \*\*matrix2 = (int \*\*)malloc(rows2 \* sizeof(int \*));

for (int i = 0; i < rows2; ++i) {

matrix2[i] = (int \*)malloc(cols2 \* sizeof(int));

}

int \*\*result = (int \*\*)malloc(rows1 \* sizeof(int \*));

for (int i = 0; i < rows1; ++i) {

result[i] = (int \*)malloc(cols2 \* sizeof(int));

}

// Input matrices

inputMatrix(matrix1, rows1, cols1);

inputMatrix(matrix2, rows2, cols2);

// Addition of matrices

printf("\n(i) Addition of Two Matrices:\n");

addMatrices(matrix1, matrix2, result, rows1, cols1);

displayMatrix(result, rows1, cols1);

// Multiplication of matrices

printf("\n(ii) Multiplication of Two Matrices:\n");

multiplyMatrices(matrix1, rows1, cols1, matrix2, cols2, result);

displayMatrix(result, rows1, cols2);

// Transpose of matrix

printf("\n(iii) Transpose of a Matrix:\n");

int \*\*transposeResult = (int \*\*)malloc(cols1 \* sizeof(int \*));

for (int i = 0; i < cols1; ++i) {

transposeResult[i] = (int \*)malloc(rows1 \* sizeof(int));

}

transposeMatrix(matrix1, rows1, cols1, transposeResult);

displayMatrix(transposeResult, cols1, rows1);

// Free allocated memory

for (int i = 0; i < rows1; ++i) {

free(matrix1[i]);

}

free(matrix1);

for (int i = 0; i < rows2; ++i) {

free(matrix2[i]);

}

free(matrix2);

for (int i = 0; i < rows1; ++i) {

free(result[i]);

}

free(result);

for (int i = 0; i < cols1; ++i) {

free(transposeResult[i]);

}

free(transposeResult);

return 0;

}

1. #include <stdio.h>

// Recursive function to find the factorial of a number

unsigned long long int factorialRecursive(int n) {

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

return 1;

} else {

return n \* factorialRecursive(n - 1);

}

}

// Non-recursive function to find the factorial of a number

unsigned long long int factorialNonRecursive(int n) {

unsigned long long int result = 1;

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

result \*= i;

}

return result;

}

int main() {

int num;

// Input

printf("Enter a non-negative integer: ");

scanf("%d", &num);

if (num < 0) {

printf("Error: Please enter a non-negative integer.\n");

return 1; // Exit with an error code

}

// Recursive factorial

printf("Factorial (Recursive): %llu\n", factorialRecursive(num));

// Non-recursive factorial

printf("Factorial (Non-Recursive): %llu\n", factorialNonRecursive(num));

return 0;

}

1. #include <stdio.h>

// Recursive function to find the GCD of two numbers

int gcdRecursive(int a, int b) {

if (b == 0) {

return a;

} else {

return gcdRecursive(b, a % b);

}

}

// Non-recursive function to find the GCD of two numbers

int gcdNonRecursive(int a, int b) {

while (b != 0) {

int temp = b;

b = a % b;

a = temp;

}

return a;

}

int main() {

int num1, num2;

// Input

printf("Enter two positive integers separated by space: ");

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

if (num1 <= 0 || num2 <= 0) {

printf("Error: Please enter positive integers.\n");

return 1; // Exit with an error code

}

// Recursive GCD

printf("GCD (Recursive): %d\n", gcdRecursive(num1, num2));

// Non-recursive GCD

printf("GCD (Non-Recursive): %d\n", gcdNonRecursive(num1, num2));

return 0;

}

1. #include <stdio.h>

// Recursive function to calculate x^n

double powerRecursive(double x, int n) {

if (n == 0) {

return 1;

} else if (n > 0) {

return x \* powerRecursive(x, n - 1);

} else {

return 1.0 / (x \* powerRecursive(x, -n - 1));

}

}

// Non-recursive function to calculate x^n

double powerNonRecursive(double x, int n) {

double result = 1.0;

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

result \*= x;

}

return (n >= 0) ? result : 1.0 / result;

}

int main() {

double base;

int exponent;

// Input

printf("Enter the base (x) and exponent (n) separated by space: ");

scanf("%lf %d", &base, &exponent);

// Recursive power

printf("Power (Recursive): %.6f\n", powerRecursive(base, exponent));

// Non-recursive power

printf("Power (Non-Recursive): %.6f\n", powerNonRecursive(base, exponent));

return 0;

}

1. #include <stdio.h>

int main() {

int n;

// Input the size of the array

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

scanf("%d", &n);

if (n <= 0) {

printf("Error: Array size should be a positive integer.\n");

return 1; // Exit with an error code

}

// Allocate memory for the array

int \*arr = (int \*)malloc(n \* sizeof(int));

// Check if memory allocation is successful

if (arr == NULL) {

printf("Error: Memory allocation failed.\n");

return 1; // Exit with an error code

}

// Input elements into the array using a pointer

printf("Enter %d integers for the array:\n", n);

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

printf("Enter element %d: ", i + 1);

scanf("%d", arr + i); // Using pointer arithmetic to access array elements

}

// Display the values using the array

printf("Array values: ");

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

printf("%d ", arr[i]); // Accessing array elements using array notation

}

printf("\n");

// Free allocated memory

free(arr);

return 0;

}

1. #include <stdio.h>

int main() {

int n;

// Input the size of the array

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

scanf("%d", &n);

if (n <= 0) {

printf("Error: Array size should be a positive integer.\n");

return 1; // Exit with an error code

}

// Allocate memory for the array

int \*arr = (int \*)malloc(n \* sizeof(int));

// Check if memory allocation is successful

if (arr == NULL) {

printf("Error: Memory allocation failed.\n");

return 1; // Exit with an error code

}

// Input elements into the array using a pointer

printf("Enter %d integers for the array:\n", n);

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

printf("Enter element %d: ", i + 1);

scanf("%d", arr + i); // Using pointer arithmetic to access array elements

}

// Display the values in reverse order using a pointer

printf("Array values in reverse order: ");

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

printf("%d ", \*(arr + i)); // Using pointer arithmetic to access array elements in reverse order

}

printf("\n");

// Free allocated memory

free(arr);

return 0;

}

1. #include <stdio.h>

int main() {

int n;

// Input the size of the array

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

scanf("%d", &n);

if (n <= 0) {

printf("Error: Array size should be a positive integer.\n");

return 1; // Exit with an error code

}

// Allocate memory for the array

int \*arr = (int \*)malloc(n \* sizeof(int));

// Check if memory allocation is successful

if (arr == NULL) {

printf("Error: Memory allocation failed.\n");

return 1; // Exit with an error code

}

// Input elements into the array using a pointer

printf("Enter %d integers for the array:\n", n);

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

printf("Enter element %d: ", i + 1);

scanf("%d", arr + i); // Using pointer arithmetic to access array elements

}

// Calculate the sum of elements using a pointer

int sum = 0;

int \*ptr = arr; // Initialize pointer to the first element of the array

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

sum += \*ptr; // Access array elements using pointer dereferencing

ptr++; // Move the pointer to the next element

}

// Display the sum

printf("Sum of %d elements: %d\n", n, sum);

// Free allocated memory

free(arr);

return 0;

}

1. #include <stdio.h>

int main() {

FILE \*file;

char filename[100];

char ch;

// Input the filename from the user

printf("Enter the filename: ");

scanf("%s", filename);

// Open the file for reading

file = fopen(filename, "r");

// Check if the file was opened successfully

if (file == NULL) {

printf("Error: Unable to open the file.\n");

return 1; // Exit with an error code

}

// Read and display the contents of the file

printf("Contents of the file '%s':\n", filename);

while ((ch = fgetc(file)) != EOF) {

putchar(ch); // Display the character to the standard output

}

// Close the file

fclose(file);

return 0;

}

1. #include <stdio.h>

int main() {

FILE \*sourceFile, \*destinationFile;

char sourceFilename[100], destinationFilename[100];

char ch;

// Input the source filename

printf("Enter the source filename: ");

scanf("%s", sourceFilename);

// Open the source file for reading

sourceFile = fopen(sourceFilename, "r");

// Check if the source file was opened successfully

if (sourceFile == NULL) {

printf("Error: Unable to open the source file.\n");

return 1; // Exit with an error code

}

// Input the destination filename

printf("Enter the destination filename: ");

scanf("%s", destinationFilename);

// Open the destination file for writing

destinationFile = fopen(destinationFilename, "w");

// Check if the destination file was opened successfully

if (destinationFile == NULL) {

fclose(sourceFile); // Close the source file

printf("Error: Unable to open the destination file.\n");

return 1; // Exit with an error code

}

// Copy contents, replacing lowercase with uppercase

while ((ch = fgetc(sourceFile)) != EOF) {

if (ch >= 'a' && ch <= 'z') {

// If the character is lowercase, convert it to uppercase

ch = ch - 'a' + 'A';

}

fputc(ch, destinationFile); // Write the character to the destination file

}

// Close the files

fclose(sourceFile);

fclose(destinationFile);

printf("File copied successfully with lowercase characters replaced by their uppercase equivalents.\n");

return 0;

}

1. #include <stdio.h>

int main(int argc, char \*argv[]) {

FILE \*file;

char filename[100], searchChar;

char ch;

int count = 0;

// Check if the correct number of command line arguments are provided

if (argc != 3) {

printf("Usage: %s <filename> <character>\n", argv[0]);

return 1; // Exit with an error code

}

// Get the filename and character from command line arguments

strcpy(filename, argv[1]);

searchChar = argv[2][0]; // Consider only the first character of the second argument

// Open the file for reading

file = fopen(filename, "r");

// Check if the file was opened successfully

if (file == NULL) {

printf("Error: Unable to open the file '%s'\n", filename);

return 1; // Exit with an error code

}

// Count the occurrences of the specified character in the file

while ((ch = fgetc(file)) != EOF) {

if (ch == searchChar) {

count++;

}

}

// Close the file

fclose(file);

// Display the result

printf("The character '%c' occurs %d times in the file '%s'\n", searchChar, count, filename);

return 0;

}

1. #include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*argv[]) {

FILE \*file;

char filename[100];

int values[10];

int index, newValue;

// Check if the correct number of command line arguments are provided

if (argc != 12) {

printf("Usage: %s <filename> <value1> <value2> ... <value10>\n", argv[0]);

return 1; // Exit with an error code

}

// Get the filename and values from command line arguments

strcpy(filename, argv[1]);

for (int i = 0; i < 10; ++i) {

values[i] = atoi(argv[i + 2]); // Convert strings to integers

}

// Open the file for writing in binary mode

file = fopen(filename, "wb");

// Check if the file was opened successfully

if (file == NULL) {

printf("Error: Unable to open the file '%s' for writing.\n", filename);

return 1; // Exit with an error code

}

// Write the values to the file

fwrite(values, sizeof(int), 10, file);

// Close the file

fclose(file);

// Ask the user for index and new value

printf("Enter the index (0 to 9) and the new value: ");

scanf("%d %d", &index, &newValue);

// Open the file for updating in binary mode

file = fopen(filename, "rb+");

// Check if the file was opened successfully

if (file == NULL) {

printf("Error: Unable to open the file '%s' for updating.\n", filename);

return 1; // Exit with an error code

}

// Move the file pointer to the specified index

fseek(file, index \* sizeof(int), SEEK\_SET);

// Write the new value at the specified index

fwrite(&newValue, sizeof(int), 1, file);

// Close the file

fclose(file);

// Open the file for reading in binary mode

file = fopen(filename, "rb");

// Check if the file was opened successfully

if (file == NULL) {

printf("Error: Unable to open the file '%s' for reading.\n", filename);

return 1; // Exit with an error code

}

// Read and print all 10 values

printf("Values in the file after modification:\n");

fread(values, sizeof(int), 10, file);

for (int i = 0; i < 10; ++i) {

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

}

printf("\n");

// Close the file

fclose(file);

return 0;

}

1. #include <stdio.h>

int main() {

FILE \*file1, \*file2, \*mergedFile;

char filename1[100], filename2[100], mergedFilename[100];

char ch;

// Input the filenames for the two files

printf("Enter the filename of the first file: ");

scanf("%s", filename1);

// Open the first file for reading

file1 = fopen(filename1, "r");

// Check if the first file was opened successfully

if (file1 == NULL) {

printf("Error: Unable to open the first file '%s'\n", filename1);

return 1; // Exit with an error code

}

// Input the filename for the second file

printf("Enter the filename of the second file: ");

scanf("%s", filename2);

// Open the second file for reading

file2 = fopen(filename2, "r");

// Check if the second file was opened successfully

if (file2 == NULL) {

fclose(file1); // Close the first file

printf("Error: Unable to open the second file '%s'\n", filename2);

return 1; // Exit with an error code

}

// Input the filename for the merged file

printf("Enter the filename of the merged file: ");

scanf("%s", mergedFilename);

// Open the merged file for writing

mergedFile = fopen(mergedFilename, "w");

// Check if the merged file was opened successfully

if (mergedFile == NULL) {

fclose(file1); // Close the first file

fclose(file2); // Close the second file

printf("Error: Unable to open the merged file '%s'\n", mergedFilename);

return 1; // Exit with an error code

}

// Copy contents of the first file to the merged file

while ((ch = fgetc(file1)) != EOF) {

fputc(ch, mergedFile);

}

// Copy contents of the second file to the merged file

while ((ch = fgetc(file2)) != EOF) {

fputc(ch, mergedFile);

}

// Close the files

fclose(file1);

fclose(file2);

fclose(mergedFile);

printf("Files merged successfully into '%s'\n", mergedFilename);

return 0;

}

1. #include <stdio.h>

#include <string.h>

int romanToDecimal(char romanNumeral[]) {

int i, decimalNum = 0;

for (i = 0; i < strlen(romanNumeral); i++) {

switch (romanNumeral[i]) {

case 'I':

if (romanNumeral[i + 1] == 'V' || romanNumeral[i + 1] == 'X')

decimalNum -= 1;

else

decimalNum += 1;

break;

case 'V':

decimalNum += 5;

break;

case 'X':

if (romanNumeral[i + 1] == 'L' || romanNumeral[i + 1] == 'C')

decimalNum -= 10;

else

decimalNum += 10;

break;

case 'L':

decimalNum += 50;

break;

default:

printf("Invalid Roman numeral character: %c\n", romanNumeral[i]);

return -1; // Return -1 for invalid input

}

}

return decimalNum;

}

int main() {

char romanNumeral[10];

// Input the Roman numeral

printf("Enter a Roman numeral (I to L): ");

scanf("%s", romanNumeral);

// Convert and display the decimal equivalent

int decimalEquivalent = romanToDecimal(romanNumeral);

if (decimalEquivalent != -1) {

printf("Decimal equivalent: %d\n", decimalEquivalent);

} else {

printf("Error: Invalid Roman numeral.\n");

}

return 0;

}

1. #include <stdio.h>

void decimalToRoman(int number) {

if (number < 1 || number > 50) {

printf("Error: Input number must be in the range 1 to 50.\n");

return;

}

char \*romanNumerals[] = {"", "I", "II", "III", "IV", "V", "VI", "VII", "VIII", "IX",

"X", "XI", "XII", "XIII", "XIV", "XV", "XVI", "XVII", "XVIII", "XIX",

"XX", "XXI", "XXII", "XXIII", "XXIV", "XXV", "XXVI", "XXVII", "XXVIII", "XXIX",

"XXX", "XXXI", "XXXII", "XXXIII", "XXXIV", "XXXV", "XXXVI", "XXXVII", "XXXVIII", "XXXIX",

"XL", "XLI", "XLII", "XLIII", "XLIV", "XLV", "XLVI", "XLVII", "XLVIII", "XLIX", "L"};

printf("Roman equivalent: %s\n", romanNumerals[number]);

}

int main() {

int number;

// Input the number

printf("Enter a number (1 to 50): ");

scanf("%d", &number);

// Convert and display the Roman equivalent

decimalToRoman(number);

return 0;

}

1. #include <stdio.h>

#include <string.h>

void insertSubstring(char mainString[], char subString[], int position) {

int mainLength = strlen(mainString);

int subLength = strlen(subString);

if (position < 0 || position > mainLength) {

printf("Error: Invalid position for insertion.\n");

return;

}

// Create a temporary array to store the result

char resultString[100];

// Copy the substring before the insertion point

strncpy(resultString, mainString, position);

// Concatenate the substring

strcat(resultString, subString);

// Concatenate the remaining part of the main string

strcat(resultString, mainString + position);

// Update the main string with the result

strcpy(mainString, resultString);

printf("After insertion: %s\n", mainString);

}

void deleteCharacters(char mainString[], int position, int n) {

int mainLength = strlen(mainString);

if (position < 0 || position >= mainLength || n < 0) {

printf("Error: Invalid position or number of characters for deletion.\n");

return;

}

// Create a temporary array to store the result

char resultString[100];

// Copy the substring before the deletion point

strncpy(resultString, mainString, position);

// Concatenate the remaining part of the main string after deletion

strcat(resultString, mainString + position + n);

// Update the main string with the result

strcpy(mainString, resultString);

printf("After deletion: %s\n", mainString);

}

int main() {

char mainString[100], subString[50];

int position, n;

// Input the main string

printf("Enter the main string: ");

gets(mainString);

// Input the substring and position for insertion

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

gets(subString);

printf("Enter the position for insertion: ");

scanf("%d", &position);

// Perform insertion operation

insertSubstring(mainString, subString, position);

// Input the position and number of characters for deletion

printf("Enter the position for deletion: ");

scanf("%d", &position);

printf("Enter the number of characters to delete: ");

scanf("%d", &n);

// Perform deletion operation

deleteCharacters(mainString, position, n);

return 0;

}

1. #include <stdio.h>

#include <string.h>

// Function to check if a string is a palindrome

int isPalindrome(char str[]) {

int length = strlen(str);

// Start from both ends of the string and compare characters

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

if (str[i] != str[j]) {

return 0; // Not a palindrome

}

}

return 1; // Palindrome

}

int main() {

char inputString[100];

// Input the string

printf("Enter a string: ");

gets(inputString);

// Check if the string is a palindrome

if (isPalindrome(inputString)) {

printf("The string is a palindrome.\n");

} else {

printf("The string is not a palindrome.\n");

}

return 0;

}

1. #include <stdio.h>

int findCharacterPosition(char S[], char ch) {

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

if (S[i] == ch) {

return i; // Return the position if character is found

}

}

return -1; // Return -1 if character is not found

}

int main() {

char inputString[100];

char searchChar;

int position;

// Input the string

printf("Enter a string: ");

gets(inputString);

// Input the character to search

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

scanf(" %c", &searchChar);

// Find and display the position of the character

position = findCharacterPosition(inputString, searchChar);

if (position != -1) {

printf("The character '%c' is found at position: %d\n", searchChar, position);

} else {

printf("The character '%c' is not found in the string.\n", searchChar);

}

return 0;

}

1. #include <stdio.h>

void countLinesWordsCharacters(char text[]) {

int lines = 0, words = 0, characters = 0;

int isWord = 0; // Flag to check if a word is being counted

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

characters++;

// Check for newline character to count lines

if (text[i] == '\n') {

lines++;

}

// Check for space, tab, or newline to identify word boundaries

if (text[i] == ' ' || text[i] == '\t' || text[i] == '\n') {

isWord = 0; // Reset the word flag

} else if (!isWord) {

isWord = 1; // Set the word flag and increment word count

words++;

}

}

printf("Lines: %d\n", lines);

printf("Words: %d\n", words);

printf("Characters: %d\n", characters);

}

int main() {

char inputText[1000];

// Input the text

printf("Enter a text (Ctrl+D to end on Unix/Linux, Ctrl+Z on Windows):\n");

// Read the text until EOF (Ctrl+D on Unix/Linux, Ctrl+Z on Windows)

while (fgets(inputText, sizeof(inputText), stdin) != NULL) {}

// Count lines, words, and characters

countLinesWordsCharacters(inputText);

return 0;

}

1. #include <stdio.h>

// Function to perform linear search

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

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

if (arr[i] == key) {

return i; // Return the index if key is found

}

}

return -1; // Return -1 if key is not found

}

int main() {

int arr[100], size, key, result;

// Input the size of the array

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

scanf("%d", &size);

// Input the elements of the array

printf("Enter %d elements:\n", size);

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

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

}

// Input the key to search

printf("Enter the key to search: ");

scanf("%d", &key);

// Perform linear search and get the result

result = linearSearch(arr, size, key);

// Display the result

if (result != -1) {

printf("Key %d found at index %d.\n", key, result);

} else {

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

}

return 0;

}

1. #include <stdio.h>

// Function to perform binary search

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

int low = 0, high = size - 1, mid;

while (low <= high) {

mid = (low + high) / 2;

if (arr[mid] == key) {

return mid; // Return the index if key is found

} else if (arr[mid] < key) {

low = mid + 1; // Search in the right half

} else {

high = mid - 1; // Search in the left half

}

}

return -1; // Return -1 if key is not found

}

int main() {

int arr[100], size, key, result;

// Input the size of the array

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

scanf("%d", &size);

// Input the sorted elements of the array

printf("Enter %d sorted elements:\n", size);

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

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

}

// Input the key to search

printf("Enter the key to search: ");

scanf("%d", &key);

// Perform binary search and get the result

result = binarySearch(arr, size, key);

// Display the result

if (result != -1) {

printf("Key %d found at index %d.\n", key, result);

} else {

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

}

return 0;

}

1. #include <stdio.h>

// Function to perform Bubble Sort

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

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

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

// Swap if the element found is greater than the next element

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

// Swap the elements

int temp = arr[j];

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

arr[j + 1] = temp;

}

}

}

}

int main() {

int arr[100], size;

// Input the size of the array

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

scanf("%d", &size);

// Input the elements of the array

printf("Enter %d elements:\n", size);

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

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

}

// Perform Bubble Sort

bubbleSort(arr, size);

// Display the sorted array

printf("Sorted array in ascending order:\n");

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

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

}

printf("\n");

return 0;

}

1. #include <stdio.h>

// Function to perform Selection Sort in descending order

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

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

int maxIndex = i;

// Find the index of the maximum element in the unsorted part

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

if (arr[j] > arr[maxIndex]) {

maxIndex = j;

}

}

// Swap the found maximum element with the first element

int temp = arr[i];

arr[i] = arr[maxIndex];

arr[maxIndex] = temp;

}

}

int main() {

int arr[100], size;

// Input the size of the array

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

scanf("%d", &size);

// Input the elements of the array

printf("Enter %d elements:\n", size);

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

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

}

// Perform Selection Sort in descending order

selectionSortDescending(arr, size);

// Display the sorted array in descending order

printf("Sorted array in descending order:\n");

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

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

}

printf("\n");

return 0;

}

1. #include <stdio.h>

// Function to perform Insertion Sort in ascending order

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

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

int key = arr[i];

int j = i - 1;

// Move elements greater than key to one position ahead

while (j >= 0 && arr[j] > key) {

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

j--;

}

arr[j + 1] = key;

}

}

int main() {

int arr[100], size;

// Input the size of the array

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

scanf("%d", &size);

// Input the elements of the array

printf("Enter %d elements:\n", size);

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

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

}

// Perform Insertion Sort in ascending order

insertionSortAscending(arr, size);

// Display the sorted array in ascending order

printf("Sorted array in ascending order:\n");

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

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

}

printf("\n");

return 0;

}

1. #include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Comparison function for qsort

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

return strcmp(\*(const char \*\*)a, \*(const char \*\*)b);

}

int main() {

char names[100][50]; // Assuming a maximum of 100 names, each with a maximum length of 50 characters

int n;

// Input the number of names

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

scanf("%d", &n);

// Input the names

printf("Enter %d names:\n", n);

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

scanf("%s", names[i]);

}

// Use qsort to sort the array of names

qsort(names, n, sizeof(names[0]), compareStrings);

// Display the sorted array of names

printf("Sorted array of names:\n");

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

printf("%s\n", names[i]);

}

return 0;

}