

**Affiliated To Anna University, Chennai & Approved by AICTE, New Delhi.**

**Eachanari, Coimbatore, Tamil Nadu,India-641021.**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (CYBER SECURITY)**

**RECORD NOTE BOOK**

### 22ES105 - PROBLEM SOLVING TECHNIQUES – I LABORATORY

NAME :

REGISTER NUMBER : YEAR/SEMESTER :

ACADEMIC YEAR :



### Affiliated To Anna University, Chennai & Approved by AICTE, New Delhi.

**Eachanari, Coimbatore, Tamil Nadu,India-641021.**

**BONAFIDE CERTIFICATE**

NAME :

ACADEMIC YEAR :

YEAR/SEMESTER :

BRANCH :

**UNIVERSITY REGISTER NUMBER: ……………………………….**

Certified that this is the Bonafide record of work done by the above student in the

Laboratory during the year 2024-2025.

**Staff-in-Charge Head of the Department**

Submitted for the Practical Examination held on

**Internal Examiner External Examiner**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S. No** | **Date** | **Experiment Name** | **Page No:** | **Marks** | **Staff Sign** |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S.No** | **Date** | **Experiment Name** | **Page No:** | **Marks** | **Staff Sign** |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**EXPT NO : DATE :**

# Calculate and display the area of rectangle.

Write a C program that calculates and displays the area of a rectangle. The program should prompt the user to enter the length and width of the rectangle, then calculate and print the area.

#### Example:

**Sample Input: 1**

5 4

#### Sample Output: 1

20

#### Sample Input: 2

12.5 7.8

#### Sample Output: 2

97.5

#### Sample Input: 3

25.55 10.2

#### Sample Output: 3

260.61

#### Input Format:

Two floating-point numbers separated by a newline character, representing the length and width of the rectangle.

#### Output Format:

A single floating-point number representing the calculated area of the rectangle.

#### Constraints:

The length and width values should be positive numbers.

#### Hint:

The area of a rectangle is calculated by multiplying its length and width: Area = length \* width.

#### Naming Conventions:

Use meaningful variable names like length, width, and area.

## Solution

#include <stdio.h> int main() {

float length, width, area; scanf("%f %f", &length, &width); area = length \* width; printf("%.2f\n", area);

return 0;

}

## Analysis:

Time Complexity

**- O(1)**

Coding Conventions

**- Beginner**

Space Complexity

**- O(1)**

Error Handling

**- Beginner**

Logic Analysis

**- Beginner**

Code Reusability

**- Beginner**

Algorithmic Analysis

**- Beginner**

Code Accuracy

**- 100%**

Code Proficiency

**- Beginner**

**EXPT NO : DATE :**

# Display your personal details.

Write a C program to prompt the user to enter their personal details, including their name, age, address, and phone number. Store these details in appropriate variables and then display them in a formatted manner.

#### Sample Input 1:

John Doe 30

123 Main Street, Anytown 555-123-4567

#### Sample Output 3:

Name: John Doe Age: 30

Address: 123 Main Street, Anytown Phone Number: 555-123-4567

#### Sample Input 2:

Jane Smith 25

456 Elm Avenue, Somecity 555-987-6543

#### Sample Output 2:

Name: Jane Smith Age: 25

Address: 456 Elm Avenue, Somecity Phone Number: 555-987-6543

#### Sample Input 3:

Peter Jones 40

789 Oak Lane, Yourtown 555-567-8901

#### Sample Output 3:

Name: Peter Jones Age: 40

Address: 789 Oak Lane, Yourtown Phone Number: 555-567-8901

### Input Format:

 The program will prompt the user to enter their name, age, address, and phone number separately.

### Output Format:

 The program should display the entered personal details in a clear and formatted manner.

### Constraints:

 Maximum length of name, address, and phone number should not exceed 50 characters.  Age should be a positive integer.

### Hint:

 Use the scanf function to read user input and the printf function to display the details.  Use %[^\n] in scanf to capture strings with spaces.

### Naming Conventions:

 Use descriptive variable names like name, age, address, and phoneNumber.

## Solution

#include <stdio.h> int main() {

int age;

char name[100],address[100],phone[13]; fgets(name,100,stdin); scanf("%d",&a

scanf("%d",&age);

getchar();

fgets(address,100,stdin); fgets(phone,13,stdin); printf("\nName: %s",name); printf("Age: %d\n",age); printf("Address: %s",address); printf("Phone Number: %s\n",phone); return 0;

}

## Analysis:

Time Complexity

**- O(1)**

Coding Conventions

**- Beginner**

Space Complexity

**- O(1)**

Error Handling

**- Beginner**

Logic Analysis

**- Beginner**

Code Reusability

**- Beginner**

Algorithmic Analysis

**- Beginner**

Code Accuracy

**- 100%**

Code Proficiency

**- Beginner**

**EXPT NO : DATE :**

# Leap year or not

Write a C program that determines if a given year is a leap year or not. A leap year occurs: \* Every 4 years \* Except for years that are divisible by 100 but not divisible by 400. For example, 2000 was a leap year (divisible by 400), but 1900 was not (divisible by 100 but not by 400).

#### Example:

**Sample Input: 1**

2024

#### Sample Output: 1

Leap Year

#### Sample Input: 2

1900

#### Sample Output: 2

Not a Leap Year

#### Sample Input: 3

2000

#### Sample Output: 3

Leap Year

#### Input Format:

An integer representing the year.

#### Output Format:

Print "Leap Year" if the year is a leap year, otherwise print "Not a Leap Year".

Constraints:

The input year will be a positive integer.

#### Hint:

Use the modulus operator (%) to check for divisibility.

#### Naming Conventions:

Variable names should be descriptive (e.g., year).

## Solution

#include <stdio.h> int main() {

int year; scanf("%d", &year);

if ((year % 400 == 0) || (year % 4 == 0 && year % 100 != 0)) {

printf("Leap Year\n");

}

else {

printf("Not a Leap Year\n");

}

return 0;

}

## Analysis:

Time Complexity

**- O(1)**

Coding Conventions

**- Beginner**

Space Complexity

**- O(1)**

Error Handling

**- Beginner**

Logic Analysis

**- Beginner**

Code Reusability

**- Beginner**

Algorithmic Analysis

**- Beginner**

Code Accuracy

**- 100%**

Code Proficiency

**- Beginner**

**EXPT NO : DATE :**

# Reverse of a given number

Write a C program that takes an integer as input and outputs the reverse of the integer.

### Example

#### Sample Input 1:

12345

#### Sample Output 1:

54321

#### Sample Input 2:

2345

#### Sample Output 2:

5432

#### Sample Input 3:

1000

#### Sample Output 3:

0001

### Input Format:

The input will be a single integer, which can be positive or negative.

### Output Format:

The output should be a single integer representing the reversed number.

### Constraints:

The input integer will be within the range of a 32-bit signed integer (-2,147,483,648 to 2,147,483,647).

The reversed number must also fall within the range of a 32-bit signed integer. If reversing causes overflow, output 0.

#### Hint:

You can reverse a number by extracting its digits one by one using modulus and division operations, and then reconstructing the reversed number. Be mindful of integer overflow.

#### Naming Conventions:

Use meaningful variable names that follow standard C naming conventions. 4o

## Solution

#include<stdio.h> #include<string.h> int main()

{

char n[100],r[100]; int len,j=0; scanf("%s",n); len=strlen(n);

for(int i=len-1;i>=0;i--)

{

r[j++]=n[i];

}

r[j]='\0';

printf("%s",r); return 0;

}

## Analysis:

Time Complexity

**- O(n)**

Coding Conventions

**- Beginner**

Space Complexity

**- O(n)**

Error Handling

**- Beginner**

Logic Analysis

**- Beginner**

Code Reusability

**- Beginner**

Algorithmic Analysis

**- Beginner**

Code Accuracy

**- 100%**

Code Proficiency

**- Beginner**

**EXPT NO : DATE :**

# Addition, subtraction, multiplication, division, modulus and square of a number.

Create a C program that acts as a basic calculator. It should be able to perform the following operations based on user input: 1. \*\*Addition:\*\* Add two numbers. 2. \*\*Subtraction:\*\* Subtract two numbers. 3. \*\*Multiplication:\*\* Multiply two numbers. 4. \*\*Division:\*\* Divide two numbers. 5.

\*\*Modulus:\*\* Find the remainder of the division of two numbers. 6. \*\*Square:\*\* Calculate the square of a number. The program should prompt the user to enter their choice of operation and the corresponding operands.

#### Example:

**Sample Input: 1**

1 5 3

#### Sample Output: 1

8

#### Sample Input: 2

4 10 3

#### Sample Output: 2

3.33

#### Sample Input: 3

6 -2.5

#### Sample Output: 3

6.25

#### Input Format:

The input will consist of multiple lines. The first line will contain an integer representing the desired operation (1-6). Subsequent lines will contain the operands (one or two depending on the operation) as floating-point numbers.

#### Output Format:

The output should be a single line displaying the result of the chosen operation. The result should be formatted appropriately based on the operation (e.g., as an integer for addition, subtraction, multiplication, modulus; as a floating-point number for division and square).

#### Constraints:

The program should handle division by zero gracefully and provide an appropriate error message. For the square operation, assume the input will be a floating-point number.

#### Hint:

Utilize a `switch` statement to efficiently handle the different calculator operations based on user input.

#### Naming Conventions:

Use descriptive variable names like 'num1', 'num2', 'operator', 'result', etc.

## Solution

#include <stdio.h> int main() {

int operation;

float num1, num2, result; scanf("%d", &operation);

switch (operation) {

case 1:

case 2:

case 3:

case 4:

printf("");

scanf("%f %f", &num1, &num2); result = num1 + num2; printf(" %.2f\n", result); break;

printf("");

scanf("%f %f", &num1, &num2); result = num1 - num2; printf("%.2f\n", result); break;

printf("");

scanf("%f %f", &num1, &num2); result = num1 \* num2; printf("%.2f\n", result); break;

printf("");

scanf("%f %f", &num1, &num2); if (num2 == 0) {

printf("Error: Division by zero!\n");

} else {

result = num1 / num2; printf(" %.2f\n", result);

case 5:

case 6:

default:

}

}

break;

printf("");

scanf("%f %f", &num1, &num2); if (num2 == 0) {

printf("Error: Division by zero!\n");

} else {

result = (int)num1 % (int)num2; printf(" %.2f\n", result);

}

break;

printf("");

scanf("%f", &num1); result = num1 \* num1; printf(" %.2f\n", result); break;

printf("Invalid operation!\n");

return 0;

}

## Analysis:

Time Complexity

**- O(1)**

Coding Conventions

**- Beginner**

Space Complexity

**- O(1)**

Error Handling

**- Beginner**

Logic Analysis

**- Beginner**

Code Reusability

**- Beginner**

Algorithmic Analysis

**- Beginner**

Code Accuracy

**- 100%**

Code Proficiency

**- Beginner**

**EXPT NO : DATE :**

# Armstrong number or not

An Armstrong number is a number that is equal to the sum of the cubes of its digits. For example, 371 is an Armstrong number because 3\*\*3 + 7\*\*3 + 1\*\*3 = 371. Your task is to write a C program that takes an integer as input and determines whether it is an Armstrong number or not. The program should print "Armstrong Number" if the input number is an Armstrong number, and "Not an Armstrong Number" otherwise.

#### Example:

**Sample Input: 1**

153

#### Sample Output: 1

Armstrong Number

#### Sample Input: 2

370

#### Sample Output: 2

Armstrong Number

#### Sample Input: 3

1634

**Sample Output: 3** Armstrong Number **Input Format:**

The input will be a single integer.

#### Output Format:

The output will be a string, either "Armstrong Number" or "Not an Armstrong Number".

#### Constraints:

The input number will be a positive integer.

#### Hint:

Calculate the sum of the cubes of the digits of the input number and compare it with the original number.

#### Naming Conventions:

Use meaningful variable names like 'originalNumber', 'sumOfCubes', 'digit', etc.

## Solution

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

int main() {

int num, originalNum, remainder, result = 0,digits=0; printf("");

scanf("%d", &num); originalNum=num; while (num != 0) {

digits++; num/=10;

}

num=originalNum; while(originalNum!=0)

{

remainder=originalNum%10; result+=pow(remainder,digits); originalNum/=10;

}

if (result == num) printf("Armstrong Number\n"); else

printf("Not an Armstrong Number\n"); return 0;

}

## Analysis:

Time Complexity

**- O(n)**

Coding Conventions

**- Beginner**

Space Complexity

**- O(1)**

Error Handling

**- Beginner**

Logic Analysis

**- Beginner**

Code Reusability

**- Beginner**

Algorithmic Analysis

**- Beginner**

Code Accuracy

**- 100%**

Code Proficiency

**- Beginner**

**EXPT NO : DATE :**

# Electricity Bill Calculation

An electricity company needs to automate its billing system. Your task is to develop a C program that calculates a customer's electricity bill based on their consumption and applies relevant charges. The program should adhere to the following specifications: \*\*Input:\*\* - Customer ID (integer) - Customer Name (string) - Units Consumed (integer) \*\*Calculation:\*\* - For the first 199 units: ₹1.20 per unit - For the next 200 units (200-399): ₹1.50 per unit - For the next 200 units (400-599): ₹1.80 per unit - For 600 units and above: ₹2.00 per unit - \*\*Surcharge:\*\* If the total bill amount exceeds ₹400, add a 15% surcharge. - \*\*Minimum Bill:\*\* The minimum bill amount should be ₹100. \*\*Output:\*\* - Customer ID - Customer Name - Units Consumed - Total Amount Payable Your program should handle various customer consumption scenarios and accurately calculate the bill amount.

#### Example:

**Sample Input: 1**

2357 jack 780

#### Sample Output: 1

Customer ID: 2357 Customer Name: jack Units Consumed: 780

Total Amount Payable: ₹1449.92

#### Sample Input: 2

5678 Jane 350

#### Sample Output: 2

Customer ID: 5678 Customer Name: Jane Units Consumed: 350

Total Amount Payable: ₹535.09

#### Sample Input: 3

9012 Alice 650

#### Sample Output: 3

Customer ID: 9012 Customer Name: Alice

Units Consumed: 650

Total Amount Payable: ₹1150.92

#### Input Format:

Input will be provided as separate lines: customer ID, customer name, and units consumed.

#### Output Format:

Output should be printed to the console with clear labels for customer ID, customer name, units consumed, and the total amount payable.

#### Constraints:

Customer ID should be a positive integer. Customer Name should be a string. Units consumed should be a non-negative integer.

#### Hint:

Use if-else statements to determine the applicable rate based on units consumed and calculate the total bill. Add a surcharge and ensure the minimum bill amount.

#### Naming Conventions:

Use descriptive variable names like customerId, customerName, unitsConsumed, totalBill, etc.

## Solution

#include <stdio.h> #include<string.h>

int main() {

int customerId, unitsConsumed; char customerName[100];

float totalAmount = 0.0; scanf("%d", &customerId); scanf("%s",customerName); scanf("%d", &unitsConsumed); if (unitsConsumed <= 199) {

totalAmount = unitsConsumed \* 1.20;

}

else if (unitsConsumed <= 399) {

totalAmount = 199 \* 1.20 + (unitsConsumed - 199) \* 1.50;

} else if (unitsConsumed <= 599) {

totalAmount = 199 \* 1.20 + 200 \* 1.50 + (unitsConsumed - 399) \* 1.80;

}

else {

totalAmount = 199 \* 1.20 + 200 \* 1.50 + 200 \* 1.80 + (unitsConsumed - 599) \*

2.00;

}

if (totalAmount > 400) {

totalAmount += totalAmount \* 0.15;

}

if (totalAmount < 100) { totalAmount = 100;

}

printf("Customer ID: %d\n", customerId); printf("Customer Name: %s\n", customerName); printf("Units Consumed: %d\n", unitsConsumed); printf("Total Amount Payable: ₹%.2f\n", totalAmount);

return 0;

}

## Analysis:

Time Complexity

**- O(1)**

Coding Conventions **- Intermediate**

Space Complexity

**- O(1)**

Error Handling

**- Beginner**

Logic Analysis

**- Beginner**

Code Reusability

**- Beginner**

Algorithmic Analysis

**- Beginner**

Code Accuracy

**- 100%**

Code Proficiency

**- Beginner**

**EXPT NO : DATE :**

# To sort an Array using Selection Sort Techniques

Implement a C program to sort an array of integers in ascending order using the Selection Sort algorithm. Selection Sort repeatedly finds the minimum element from the unsorted part and places it at the beginning. Your program should efficiently iterate through the array, compare elements, and swap them to achieve the sorted order.

#### Example:

**Sample Input: 1**

5 4 2 7 1 3

#### Sample Output: 1

1 2 3 4 7

#### Sample Input: 2

8 -5 12 0 8 -2 10 5 3

#### Sample Output: 2

-5 -2 0 3 5 8 10 12

#### Sample Input: 3

10 9 1 4 -3 8 2 6 5 -7 0

#### Sample Output: 3

-7 -3 0 1 2 4 5 6 8 9

#### Input Format:

The first line contains an integer 'n' representing the size of the array. The second line contains 'n' space-separated integers, which are the elements of the array.

#### Output Format:

Print the sorted array elements in ascending order, separated by spaces.

#### Constraints:

1 <= Array size <= 1000, -1000 <= Array elements <= 1000

#### Hint:

Use nested loops: the outer loop iterates through the sorted subarray, and the inner loop finds the index of the minimum element in the remaining unsorted subarray.

#### Naming Conventions:

Variable names should be descriptive (e.g., 'arraySize', 'minIndex', 'temp').

## Solution

#include <stdio.h>

void selectionSort(int array[], int arraySize) { int i, j, minIndex, temp;

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

for (j = i + 1; j < arraySize; j++) { if (array[j] < array[minIndex]) {

minIndex = j;

}

}

if (minIndex != i) { temp = array[i];

array[i] = array[minIndex]; array[minIndex] = temp;

}

}

}

int main() {

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

for (i = 0; i < arraySize; i++) { scanf("%d", &array[i]);

}

selectionSort(array, arraySize); for (i = 0; i < arraySize; i++) {

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

}

return 0;

}

## Analysis:

Time Complexity **- O(n^2)**

Coding Conventions **- Intermediate**

Space Complexity

**- O(1)**

Error Handling

**- Beginner**

Logic Analysis

**- Beginner**

Code Reusability

**- Beginner**

Algorithmic Analysis

**- Beginner**

Code Accuracy

**- 100%**

Code Proficiency

**- Beginner**

**EXPT NO : DATE :**

# Find a element using Linear Search Techniques

You are tasked with implementing a linear search algorithm in C to find the position of a given element within a sorted array. Linear search, also known as sequential search, is a simple searching algorithm that checks each element in the array one by one until it finds the target element or reaches the end of the array. Your program should take an array of integers, the size of the array, and the element to search for as input. If the element is found in the array, it should return the index (position) of the element. Otherwise, it should return -1 to indicate that the element is not present in the array.

#### Example:

**Sample Input: 1**

5 10 20 30 40 50 30

#### Sample Output: 1

2

#### Sample Input: 2

7 2 5 8 12 16 23 38 16

#### Sample Output: 2

4

#### Sample Input: 3

10 1 3 7 10 15 21 28 36 45 55 29

#### Sample Output: 3

-1

#### Input Format:

The first line of input contains an integer representing the size of the array. The second line contains space-separated integers representing the elements of the array. The third line contains an integer representing the element to search for.

#### Output Format:

Print a single integer representing the index of the target element in the array. If the element is not found, print -1.

#### Constraints:

The size of the array will be a positive integer less than or equal to 100. The elements of the array and the target element will be integers within the range of -1000 to 1000.

#### Hint:

Traverse the array element by element and compare each element with the target element. If a match is found, return the index of the current element.

#### Naming Conventions:

Use meaningful variable names such as 'array', 'size', 'target', and 'index'.

## Solution

#include <stdio.h>

int linearSearch(int array[], int size, int target) { for (int i = 0; i < size; i++) {

if (array[i] == target) { return i;

}

}

return -1;

}

int main() {

int size, target; scanf("%d", &size); int array[size];

for (int i = 0; i < size; i++) { scanf("%d", &array[i]);

}

scanf("%d", &target);

int result = linearSearch(array, size, target); printf("%d\n", result);

return 0;

}

## Analysis:

Time Complexity

**- O(n)**

Coding Conventions

**- Beginner**

Space Complexity

**- O(1)**

Error Handling

**- Beginner**

Logic Analysis

**- Beginner**

Code Reusability

**- Beginner**

Algorithmic Analysis

**- Beginner**

Code Accuracy

**- 100%**

Code Proficiency

**- Beginner**

**EXPT NO : DATE :**

# Binary Search Techniques using Recursion.

Write a recursive function that implements the Binary Search algorithm to find the index of a target element in a sorted array. If the target element is not present in the array, return "Element not found in the array."

#### Example:

**Sample Input: 1**

5 2 4 6 8 10 6

#### Sample Output: 1

2

#### Sample Input: 2

7 1 3 5 7 9 11 13 4

#### Sample Output: 2

Element not found in the array.

#### Sample Input: 3

8 -5 -2 0 3 8 11 15 20 15

#### Sample Output: 3

6

#### Input Format:

The first line contains an integer N representing the number of elements in the array. The second line contains N space-separated integers representing the sorted array. The third line contains an integer, target, representing the element to be searched.

#### Output Format:

Return a single integer: the index of the target element if found, otherwise -1.

#### Constraints:

1 <= array.length <= 10^4, -10^4 <= array[i], target <= 10^4, array is sorted in ascending order

#### Hint:

Divide and conquer: compare the target with the middle element, then recursively search the left or right half.

#### Naming Conventions:

Variables: array (input array), target (element to search for), low (lower bound index), high (upper bound index)

## Solution

#include <stdio.h>

int binarySearch(int array[], int low, int high, int target) { if (low > high) {

return -1;

}

int mid = low + (high - low) / 2; if (array[mid] == target) {

return mid;

} else if (array[mid] > target) {

return binarySearch(array, low, mid - 1, target);

} else {

return binarySearch(array, mid + 1, high, target);

}

}

int main() { int N, target; scanf("%d", &N); int array[N];

for (int i = 0; i < N; i++) { scanf("%d", &array[i]);

}

scanf("%d", &target);

int result = binarySearch(array, 0, N - 1, target); if (result == -1) {

printf("Element not found in the array.\n");

} else {

printf("%d\n", result);

}

return 0;

}

## Analysis:

Time Complexity

**- O(log n)**

Coding Conventions

**- Beginner**

Space Complexity

**- O(1)**

Error Handling

**- Beginner**

Logic Analysis

**- Intermediate**

Code Reusability

**- Beginner**

Algorithmic Analysis **- Intermediate**

Code Accuracy

**- 100%**

Code Proficiency

**- Intermediate**

**EXPT NO : DATE :**

# Factorial of a given number using Recursion.

Write a C program to find the factorial of a given non-negative integer using recursion. The factorial of a non-negative integer 'n', denoted by n!, is the product of all positive integers less than or equal to 'n'. For example, 5! = 5 \* 4 \* 3 \* 2 \* 1 = 120. Your program should handle the case where the input is 0, as 0! is defined to be 1.

#### Example:

**Sample Input: 1**

5

#### Sample Output: 1

5 = 120

#### Sample Input: 2

1

#### Sample Output: 2

1 = 1

#### Sample Input: 3

7

#### Sample Output: 3

7 = 5040

#### Input Format:

The input will be a single integer.

#### Output Format:

The output will be a single integer representing the factorial of the input number.

#### Constraints:

The input number will be a non-negative integer.

#### Hint:

You can define a recursive function that calls itself with a decremented value until it reaches the base case (n == 0).

#### Naming Conventions:

Use descriptive variable names like 'number' and 'factorial'.

## Solution

#include <stdio.h> int factorial(int n) { if (n == 0) {

return 1;

}

else{

return n \* factorial(n-1);

}

}

int main() { int number;

scanf("%d", &number); if (number >= 0){

printf("%d = %d\n",number,factorial(number));

}else {

printf("the factorial cannot be found for negative numbers");

}

return 0;

}

## Analysis:

Time Complexity

**- O(n)**

Coding Conventions

**- Beginner**

Space Complexity

**- O(n)**

Error Handling

**- Beginner**

Logic Analysis

**- Beginner**

Code Reusability

**- Beginner**

Algorithmic Analysis

**- Beginner**

Code Accuracy

**- 100%**

Code Proficiency

**- Beginner**

**EXPT NO : DATE :**

# Matrix Multiplication

Write a C program that takes two matrices as input from the user and calculates their matrix product. The program should first prompt the user to enter the dimensions of the two matrices (rows and columns) and then the elements of each matrix. After performing the matrix multiplication, the program should neatly display the resulting product matrix.

#### Example:

**Sample Input: 1**

2 2 2 2 1 2 3 4 1 2 3 4

#### Sample Output: 1

7 10

15 22

#### Sample Input: 2

2 3 3 2 2 1 3 1 4 2 1 0 2 1 4 3

#### Sample Output: 2

16 10

17 10

#### Sample Input: 3

3 2 2 3 1 2 1 -2 3 2 2 1 0 0 -1 2 1 1 3

#### Sample Output: 3

2 -1 4

2 3 -4

6 1 4

#### Input Format:

The input will be provided as follows: Number of rows of matrix 1, number of columns of matrix 1, number of rows of matrix 2, number of columns of matrix 2, followed by the elements of matrix 1 row-wise and then the elements of matrix 2 row-wise.

#### Output Format:

The output should be a matrix printed row-wise, with each row on a new line and elements within a row separated by spaces.

#### Constraints:

1) The number of columns in the first matrix must be equal to the number of rows in the second matrix for matrix multiplication to be valid. 2) The maximum dimensions of the matrices should be reasonable to prevent potential memory issues.

#### Hint:

Use nested loops to iterate through the rows and columns of the matrices, and perform the dot product of corresponding rows and columns to calculate the elements of the product matrix.

#### Naming Conventions:

Use meaningful variable names like matrix1, matrix2, rows1, cols1, rows2, cols2, productMatrix, etc.

## Solution

#include <stdio.h> int main() {

int r1,r2,c1,c2; scanf("%d",&r1);

scanf("%d",&c1);

scanf("%d",&r2);

scanf("%d",&c2);

int A[r1][c1],B[r2][c2],C[r1][c2];

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

{

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

{

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

}

}

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

{

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

{

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

}

}

if(c1==r2)

{

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

{

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

{

C[i][j]=0;

}

}

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

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

{

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

{

C[i][j]+=A[i][k]\*B[k][j];

}

}

}

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

{

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

{

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

}

printf("\n");

}

}

return 0;

}

## Analysis:

Time Complexity **- O(r1 \* c1 \* c2)**

Coding Conventions

**- Beginner**

Space Complexity

**- O(r1 \* c2)**

Error Handling

**- Beginner**

Logic Analysis

**- Intermediate**

Code Reusability

**- Beginner**

Algorithmic Analysis **- Intermediate**

Code Accuracy

**- 100%**

Code Proficiency

**- Intermediate**

**EXPT NO : DATE :**

# String Operations (Using Predefined Functions)

Your task is to write a robust C program that demonstrates a comprehensive understanding of string manipulation. This program should utilize pre-defined string functions to perform the following operations: 1. \*\*String Length:\*\* Calculate and display the length of a given string. 2. \*\*String Copy:\*\* Create a copy of a given string into another string variable. 3. \*\*String Compare:\*\* Compare two strings for equality and indicate whether they are identical or not. 4. \*\*String Reverse:\*\* Reverse the order of characters in a given string. 5. \*\*String Lower:\*\* Convert all characters in a string to lowercase. 6. \*\*String Upper:\*\* Convert all characters in a string to uppercase. 7. \*\*String Concatenation:\*\* Combine two strings together to form a new string. Your program should be well- structured, properly commented, and handle user input gracefully.

#### Example:

**Sample Input: 1**

Coding Challenges Fun with Strings

#### Sample Output: 1

Length of string: 17

Copied string: Coding Challenges

Enter another string for comparison: Comparing 'Coding Challenges' and 'Fun with Strings': First string is smaller

Reversed string: segnellahC gnidoC Lowercase string: coding challenges Uppercase string: CODING CHALLENGES

Concatenated string: Coding Challenges Fun with Strings

#### Sample Input: 2

Programming Code

#### Sample Output: 2

Length of string: 11

Copied string: Programming

Enter another string for comparison: Comparing 'Programming' and 'Code': First string is greater Reversed string: gnimmargorP

Lowercase string: programming Uppercase string: PROGRAMMING C

Concatenated string: Programming Code

#### Sample Input: 3

abc xyz

#### Sample Output: 3

Length of string: 3 Copied string: abc

Enter another string for comparison: Comparing 'abc' and 'xyz': First string is smaller Reversed string: cba

Lowercase string: abc Uppercase string: ABC Concatenated string: abc xyz

#### Input Format:

The input will consist of a single line containing the string.

#### Output Format:

The output should display the results of each string operation on separate lines.

#### Constraints:

Maximum input string length is limited to 100 characters.

#### Hint:

Explore the string.h library in C for pre-defined functions related to string manipulation. Understand the purpose and usage of functions like strlen, strcpy, strcmp, strrev, strlwr, strupr, and strcat.

Implement these functions to build your string operations program.

#### Naming Conventions:

Use descriptive variable names (e.g., inputString, copiedString) that clearly indicate their purpose.

## Solution

#include <stdio.h> #include <string.h> #include <ctype.h>

void strrev(char str[]){ int n=strlen(str); for(int i=0;i<n/2;i++)

{

char temp=str[i]; str[i]=str[n-i-1];

str[n-i-1]=temp;

}

}

void strlwr(char str[])

{

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

{

str[i]=tolower(str[i]);

}

}

void strupr(char str[])

{

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

{

str[i]=toupper(str[i]);

}

}

int main()

{

char str1[100],str2[100],copiedStr[100]; gets(str1);

int length=strlen(str1);

printf("Length of string: %d\n",length); strcpy(copiedStr,str1);

printf("Copied string: %s\n",copiedStr); printf("Enter another string for comparison: "); gets(str2);

int comparisonResult=strcmp(str1,str2); if(comparisonResult==0){

printf("Comparing '%s' and '%s': Both strings are equal\n",str1,str2);

}

else if(comparisonResult>0)

{

printf("Comparing '%s' and '%s': First string is greater\n",str1,str2);

}

else{

printf("Comparing '%s' and '%s': First string is smaller\n",str1,str2);

}

strrev(str1);

printf("Reversed string: %s\n",str1); strrev(str1);

strlwr(str1);

printf("Lowercase string: %s\n",str1); strupr(str1);

printf("Uppercase string: %s\n",str1); strcpy(str1,copiedStr);

strcat(str1," "); strcat(str1,str2);

printf("Concatenated string: %s\n",str1); return 0;

}

## Analysis:

Time Complexity

**- O(n)**

Coding Conventions

**- Beginner**

Space Complexity

**- O(n)**

Error Handling

**- Beginner**

Logic Analysis

**- Beginner**

Code Reusability

**- Beginner**

Algorithmic Analysis

**- Beginner**

Code Accuracy

**- 100%**

Code Proficiency

**- Beginner**

**EXPT NO : DATE :**

# String Operations (Without Built- in Functions)

Your task is to implement a C program that performs various string operations without using any built- in string functions (like `strlen`, `strcpy`, `strcmp`, etc.). You need to write your own functions for the following operations: \* \*\*String Length:\*\* Calculate the length of a string. \* \*\*String Copy:\*\* Copy the contents of one string to another. \* \*\*String Compare:\*\* Compare two strings for equality or lexicographic order. \* \*\*String Reverse:\*\* Reverse the characters in a string. \* \*\*String Lower:\*\* Convert all uppercase characters in a string to lowercase. \* \*\*String Upper:\*\* Convert all lowercase characters in a string to uppercase. \* \*\*String Concatenation:\*\* Append one string to the end of another string. Your program should take input strings from the user and demonstrate the correct functionality of each implemented operation.

#### Example:

**Sample Input: 1**

Hello World

#### Sample Output: 1

Length of string: 5 Copied string: Hello

Enter the second string for comparison: Comparing 'Hello' and 'World': First string is smaller Reversed string: olleH

Lowercase string: hello Uppercase string: HELLO Concatenated string: Hello World

#### Sample Input: 2

Programming is Fun

#### Sample Output: 2

Length of string: 11

Copied string: Programming

Enter the second string for comparison: Comparing 'Programming' and 'is Fun': First string is smaller Reversed string: gnimmargorP

Lowercase string: programming Uppercase string: PROGRAMMING Concatenated string: Programming is Fun

#### Sample Input: 3

Coding Challenges

#### Sample Output: 3

Length of string: 6 Copied string: Coding

Enter the second string for comparison: Comparing 'Coding' and 'Challenges': First string is greater Reversed string: gnidoC

Lowercase string: coding Uppercase string: CODING

Concatenated string: Coding Challenges

#### Input Format:

Input will consist of strings provided by the user, one string at a time.

#### Output Format:

Output should clearly display the results of each string operation performed.

#### Constraints:

Maximum string length can be assumed to be 100 characters.

#### Hint:

You can iterate through the characters of a string using array indexing and pointer arithmetic. Remember that strings in C are null-terminated character arrays.

#### Naming Conventions:

Use descriptive variable names for better code readability (e.g., `sourceString`, `destinationString`,

`stringLength`).

## Solution

#include <stdio.h>

int stringLength(const char \*str) { int length = 0;

while (str[length] != '\0') { length++;

}

return length;

}

void stringCopy(char \*destination, char \*source) { int i = 0;

while (source[i] != '\0') { destination[i] = source[i]; i++;

}

destination[i] = '\0';

}

int stringCompare(const char \*str1, const char \*str2) { while (\*str1 != '\0' && \*str2 != '\0') {

if (\*str1 != \*str2) { return \*str1-\*str2;

}

str1++; str2++;

}

return \*str1-\*str2;

}

void stringReverse(char \*str) {

int start = 0, end = stringLength(str) - 1; while (start < end) {

char temp = str[start]; str[start] = str[end]; str[end] = temp; start++;

end--;

}

}

void stringLower(char \*str) { int i = 0;

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

if (str[i] >= 'A' && str[i] <= 'Z') { str[i] += 32;

} i++;

}

}

void stringUpper(char str[]) { int i = 0;

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

if (str[i] >= 'a' && str[i] <= 'z') { str[i] -= 32;

} i++;

}

}

void stringConcatenate(char \*destination, char \*source,char \*result ) { int i = 0;

while (destination[i] != '\0') { result[i]=destination[i]; i++;

}

result[i++]=' '; int j = 0;

while (source[j] != '\0') { result[i] = source[j]; i++;

j++;

}

result[i] = '\0';

}

int main() {

char sourceString[101], destinationString[101],result[101]; scanf("%100s",sourceString);

printf("Length of string: %d\n", stringLength(sourceString)); stringCopy(destinationString, sourceString);

printf("Copied string: %s\n", destinationString); char secondString[101];

printf("Enter the second string for comparison: "); getchar();

fgets(secondString, sizeof(secondString), stdin);

int comparisonResult = stringCompare(sourceString, secondString); if (comparisonResult == 0) {

printf("Comparing '%s' and '%s': Strings are equal\n", sourceString, secondString);

} else if (comparisonResult < 0) {

printf("Comparing '%s' and '%s': First string is smaller\n", sourceString, secondString);

} else {

printf("Comparing '%s' and '%s': First string is greater\n", sourceString, secondString);

}

stringReverse(destinationString);

printf("Reversed string: %s\n", destinationString); stringReverse(destinationString); stringLower(destinationString);

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

stringUpper(destinationString);

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

stringConcatenate(sourceString, secondString,result); printf("Concatenated string: %s\n", result);

return 0;

}

## Analysis:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Time Complexity | **- O(n)** |  | Coding Conventions | **- Intermediate** |
|  |  |  |  |  |
| Space Complexity | **- O(n)** |  | Error Handling | **- Beginner** |

Logic Analysis

**- Intermediate**

Code Reusability

**- Beginner**

Algorithmic Analysis **- Intermediate**

Code Accuracy

**- 100%**

Code Proficiency

**- Intermediate**

**EXPT NO : DATE :**

# Swapping Integers

Write a C program that takes two integer values as input and swaps their values using both call by value and call by reference methods. Demonstrate the difference in behavior between the two approaches.

#### Example:

**Sample Input: 1**

5 10

#### Sample Output: 1

After swapping by value: a = 5, b = 10 After swapping by reference: a = 10, b = 5

#### Sample Input: 2

-20 30

#### Sample Output: 2

After swapping by value: a = -20, b = 30 After swapping by reference: a = 30, b = -20

#### Sample Input: 3

100000 0

#### Sample Output: 3

After swapping by value: a = 100000, b = 0 After swapping by reference: a = 0, b = 100000

#### Input Format:

Two space-separated integers on a single line.

#### Output Format:

Two lines of output: First line - values of integers after call by value, Second line - values of integers after call by reference.

#### Constraints:

Input integers should be within the valid range of the 'int' data type in C.

#### Hint:

Use pointers and dereference operator (\*) to access and modify values at memory addresses for call by reference.

#### Naming Conventions:

Use meaningful variable names like 'a', 'b' for integers, and 'temp' for temporary variable.

## Solution

#include <stdio.h>

void swapByValue(int a, int b)

{

int temp=a; a=b; b=temp;

}

void swapByReference(int \*a,int \*b)

{

int temp=\*a;

\*a=\*b;

\*b=temp;

}

int main()

{

int a,b;

scanf("%d %d",&a,&b);

printf("After swapping by value: a = %d, b = %d\n",a,b); swapByReference(&a,&b);

printf("After swapping by reference: a = %d, b = %d\n",a,b); return 0;

}

## Analysis:

Time Complexity

**- O(1)**

Coding Conventions **- Intermediate**

Space Complexity

**- O(1)**

Error Handling

**- Beginner**

Logic Analysis

**- Beginner**

Code Reusability

**- Beginner**

Algorithmic Analysis

**- Beginner**

Code Accuracy

**- 100%**

Code Proficiency

**- Beginner**

**EXPT NO : DATE :**

# Average of Array Elements (Using Pointers)

Write a C program that takes the size of an array and array elements as input from the user. Then, calculate and print the average of all the array elements using pointers. You need to accomplish this without directly using array indexing (like arr[i]). Instead, manipulate array elements exclusively through pointer arithmetic.

#### Example:

**Sample Input: 1**

5 1 2 3 4 5

#### Sample Output: 1

Average: 3.00

#### Sample Input: 2

8 10 20 5 15 25 30 12 18

#### Sample Output: 2

Average: 16.88

#### Sample Input: 3

6 -2 5 8 -10 12 0

#### Sample Output: 3

Average: 2.17

#### Input Format:

The first line contains an integer representing the size of the array. The second line contains space- separated integers, representing the array elements.

#### Output Format:

Print the average of the array elements, rounded to two decimal places, in the format 'Average: '

#### Constraints:

1 <= Array size <= 100, -1000 <= Array elements <= 1000

#### Hint:

Declare a pointer to the array. Use the pointer to traverse the array and calculate the sum of elements. Remember that \*(pointer + i) gives the value at the i-th position from the pointer's current location.

#### Naming Conventions:

Variable names should be descriptive (e.g., 'size' for array size, 'sum' for the sum of elements).

## Solution

#include <stdio.h> int main() {

int size;

float sum = 0.0; scanf("%d", &size); int arr[size];

for(int i = 0; i < size; i++) { scanf("%d", &arr[i]);

}

int \*ptr = arr;

for(int i = 0; i < size; i++) { sum += \*(ptr + i);

}

float average = sum / size; printf("Average: %.2f\n", average);

return 0;

}

## Analysis:

****

Time Complexity

**- O(n)**

Coding Conventions

**- Beginner**

Space Complexity

**- O(1)**

Error Handling

**- Beginner**

Logic Analysis

**- Beginner**

Code Reusability

**- Beginner**

Algorithmic Analysis

**- Beginner**

Code Accuracy

**- 100%**

Code Proficiency

**- Beginner**

**EXPT NO : DATE :**

# Student Details (Using Structures)

You are tasked with creating a program that manages student records. You need to define a structure named 'student' to store the following information: roll number (integer), name (string), and marks in three subjects (floating-point numbers). Your program should prompt the user to enter these details for one student, store them in the structure, and then display the entered information in a clear format.

#### Example: Sample Input: 1

1

John Doe 80.5

75.0

90.0

#### Sample Output: 1

Roll No: 1 Name: John Doe

Marks in Subject 1: 80.5

Marks in Subject 2: 75.0

Marks in Subject 3: 90.0

#### Sample Input: 2

25

Jane Smith 70.0

85.5

92.3

#### Sample Output: 2

Roll No: 25 Name: Jane Smith

Marks in Subject 1: 70.0

Marks in Subject 2: 85.5

Marks in Subject 3: 92.3

#### Sample Input: 3

100

Peter Jones

95.8

92.1

88.7

#### Sample Output: 3

Roll No: 100 Name: Peter Jones

Marks in Subject 1: 95.8

Marks in Subject 2: 92.1

Marks in Subject 3: 88.7

#### Input Format:

The input will be entered by the user on separate lines: roll number, name, marks in subject 1, marks in subject 2, and marks in subject 3.

#### Output Format:

The output should display the student's details in the following format: Roll No: [rollNo] Name: [name] Marks in Subject 1: [marks1] Marks in Subject 2: [marks2] Marks in Subject 3: [marks3]

#### Constraints:

Roll number should be a positive integer. Marks should be floating-point numbers between 0 and 100.

#### Hint:

Use `scanf` to read input, store it in structure members, and then use `printf` to display the structure data.

#### Naming Conventions:

Use meaningful variable names like 'rollNo', 'name', 'marks1', etc.

## Solution

#include <stdio.h> #include <string.h> struct Student {

int rollno; char name[100]; float marks[3];

}S1;

int main() {

scanf("%d", &S1.rollno);

scanf(" %[^\n]",S1.name); for(int j=0;j<3;j++)

{

scanf("%f", &S1.marks[j]);

}

printf("Roll No: %d\n",S1.rollno);

printf("Name: %s\n",S1.name);; for(int j=0;j<3;j++)

{

printf("Marks in Subject %d: %.1f\n",j+1, S1.marks[j]);

}

return 0;

}

## Analysis:

Time Complexity

**- O(1)**

Coding Conventions

**- Beginner**

Space Complexity

**- O(1)**

Error Handling

**- Beginner**

Logic Analysis

**- Beginner**

Code Reusability

**- Beginner**

Algorithmic Analysis

**- Beginner**

Code Accuracy

**- 100%**

Code Proficiency

**- Beginner**

****

**EXPTNO:**

# DATE : Read Data From a Text File in C

**AIM:**

To write a C program that reads data from a text file and displays it on the screen.

**ALGORITHM:**

STEP 1: Start the Program

STEP 2: Declare Variables and Open File STEP 3: Check File Status

STEP 4: Read File Contents STEP 5: Print File Contents STEP 6: Close File and Exit

**PROGRAM:**

#include<stdio.h> int main(){

FILE \*fptr;

fptr = fopen( "newfile.txt","w"); fputs("welcome to c programming. \n", fptr); fclose(fptr);

fptr = fopen( "newfile.txt", "r"); char content[1000];

while (fgets(content, sizeof(content), fptr)) { printf("%s", content);

}

fclose(fptr);

return 0;

}

**OUTPUT:**

**RESULT:**

**EXPTNO: Read Data From a Text File And**

**DATE: Append to Another File**

**AIM:**

To write a C program that reads data from a text file and appends it to the end of another

text file.

**ALGORITHM:**

STEP 1: Open Source File STEP 2: Open Destination File STEP 3: Read and Append Data STEP 4: Close Files

STEP 5: Display Appended Data

**PROGRAM:**

#include<stdio.h>

int main(){

FILE \*dest, \*source;

char app[1000];

source = fopen("newfile.txt","r");

dest = fopen("newfile1.txt","a");

while (fgets(app, sizeof(app), source))

{

fputs(app, dest);

}

printf("Data has been successfully appended to the target file.\n");

fclose(dest); fclose(source);

dest = fopen( "newfile1.txt", "r");

char disp[1000];

while (fgets(disp, sizeof(disp), dest)) { printf("%s", disp);

}

fclose(dest);

return 0;

}

**OUTPUT:**

**RESULT:**