**Register No:** RTC2024BEC023

# Index

**Register No:** RTC2024BEC023

| Name of the Experiment | Date | Page | Signature |
|---|---|---|---|
| **String Operations (Without Built-in Functions)** | | | |

**Register No:** RTC2024BEC023

# Index

**Register No:** RTC2024BEC023

**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];
 printf("");
 fgets(n,sizeof(n),stdin);
 int l=strlen(n);
 if(n[l-1]=='\n'){
 n[l-1]='\0';
 l--;
 }
 printf("");
 for(int i=l-1;i>=0;i--){
 printf("%c",n[i]);
 }
 printf("\n");
 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%** |

**Register No:** RTC2024BEC023

| Code Proficiency | **- Beginner** |
|---|---|

**Register No:** RTC2024BEC023

**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>
int main() {
 int n,minIndex,i,j,temp;
 scanf("%d",&n);
 int arr[n];
 for(int i=0;i<n;i++){
 scanf("%d",&arr[i]);
 }
 for(int i=0;i<n-1;i++){
 minIndex=i;
 for(int j=i+1;j<n;j++){
 if(arr[j] < arr[minIndex]){
 minIndex=j;
 }
 }
 temp=arr[minIndex];
 arr[minIndex]=arr[i];
 arr[i]=temp;
 }
for(int d=0;d<n;d++){
 printf("%d ",arr[d]);
}
 return 0;
}
```

# Analysis:

| | |
|---|---|
| Time Complexity | **- O(n^2)** |
| Space Complexity | **- O(1)** |
| Coding Conventions | **- Beginner** |
| Error Handling | **- Beginner** |

**Register No:** RTC2024BEC023

| | |
|---|---|
| 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 %d %d %d",&r1,&r2,&c1,&c2);
 if(c1!=r2){
 printf("Matrix Multiplication not possible");
 return 0;
 }
 int matrix1[20][20],matrix2[20][20];
 int result[20][20];
 for(int i=0;i<r1;i++){
 for(int j=0;j<c1;j++){
 scanf("%d",&matrix1[i][j]);
 }
 }
 for(int i=0;i<r2;i++){
 for(int j=0;j<c2;j++){
 scanf("%d",&matrix2[i][j]);
 }
 }
 for(int i=0;i<r1;i++){
 for(int j=0;j<c2;j++){
 result[i][j]=0;
 }
 }
 for(int i=0;i<r1;i++){
 for(int j=0;j<c2;j++){
 for(int k=0;k<c1;k++){
 result[i][j] += matrix1[i][k] * matrix2[k][j];
 }
 }
 }
  for(int i=0;i<r1;i++){
 for(int j=0;j<c2;j++){
```

```
 printf("%d ",result[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 - **Beginner** | Code Reusability - **Beginner** |
| Algorithmic Analysis - **Beginner** | Code Accuracy - **100%** |
| Code Proficiency - **Beginner** | |

**Register No:** RTC2024BEC023

**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>
int main() {
 int id,units;
 char name[51];
 float charge_per_units=0,total_bill,subcharge=0;
 scanf("%d",&id);
 scanf("%s",name);
 scanf("%d",&units);
 if(units<=199){
 total_bill = units * 1.20;
 }
else if(units<=399){
 total_bill = 199*1.20+(units - 199)*1.50;
}
else if(units<=599){
 total_bill = 199*1.20+200*1.5+(units-399)*1.80;
}
else if(units>=600){
 total_bill = 199*1.20+200*1.5+200*1.8+(units - 599)*2.00;
}
if (total_bill>400){
 subcharge=total_bill*0.15;
}
total_bill+=subcharge;
if(total_bill<100){
```

```
 total_bill=100;
}
printf("Customer ID: %d\n",id);
printf("Customer Name: %s\n",name);
printf("Units Consumed: %d\n",units);
printf("Total Amount Payable: ₹%0.2f\n",total_bill);
 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** | | |

**Register No:** RTC2024BEC023

**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
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>
// Function to reverse a string
void strRev(char *str, int l) {
 l = l - 1;
 char temp[50]; // Temporary array for reversed string
 int i = 0;
 while (str[i] != '\0') {
  temp[i] = str[l - i]; // Reverse string
```

```
 i++;
 }
 temp[i] = '\0'; // Null-terminate the reversed string
 printf("Reversed string: %s\n", temp);
 }
 // Function to convert string to lowercase
 void strLwr(char *str) {
 int i = 0;
 while (str[i] != '\0') {
 if (str[i] >= 'A' &&
str[i] <= 'Z') {
 str[i] +=
32; // Convert to lowercase
 }

i++;

}

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

}

// Function to convert string to uppercase

void strUpr(char *str) {

int i = 0;

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

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

str[i] -= 32; // Convert to uppercase

}

i++;

}

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

}

int main() {

char str1[100], str2[100], str3[100], str4[200] = ""; // Initialize str4 to avoid
undefined behavior

// Reading strings

scanf("%[^\n]s", str1); // Read the first string with spaces

getchar(); // To consume the newline character after the first input

scanf("%[^\n]s", str2); // Read the second string

int length = strlen(str1);

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

```
// Copy str1 to str3

strcpy(str3, str1);

printf("Copied string: %s\n", str3);

// Compare two strings

printf("Enter another string for comparison: Comparing '%s' and '%s': ", str1,
str2);

int cmp = strcmp(str1, str2);

if (cmp == 0) {

printf("Both strings are equal\n");

}

if (cmp < 0) {

printf("First string is smaller\n");

}

if (cmp > 0) {

printf("First string is greater\n");

}

// Reverse the string

strcpy(str3, str1);

strRev(str3, length);

// Convert the string to lowercase

strcpy(str3, str1);

strLwr(str3);
// Convert the string to uppercase

strcpy(str3, str1);

strUpr(str3);

// Concatenate the strings

strcpy(str4, str1); // Copy str1 to str4

strcat(str4, " "); // Add a space between the strings

strcat(str4, str2); // Concatenate str2 to str4

printf("Concatenated string: %s\n", str4);

return 0;
```

```
}
```

# Analysis:

| | | | |
|---|---|---|---|
| Time Complexity | **- O(n)** | Coding Conventions | **- Beginner** |
| 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 :**

# 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() {
 char name[201],address[201],num[15];
 int age;
 scanf("\n");
 scanf("%[^\n]s",name);
 scanf("%d",&age);
  scanf("\n");
```

```
 scanf("%[^\n]s",address);
 scanf("%s",num);
 printf("\n");
 printf("Name: %s\n",name);
 printf("Age: %d\n",age);
 printf("Address: %s\n",address);
 printf("Phone Number: %s\n",num);
 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** | | |

**Register No:** RTC2024BEC023

**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 binarySearchRecursive(int arr[],int low,int high,int key){
 if (low>high){
 return -1;
 }
 int mid=low+(high-low)/2;
 if (arr[mid]==key){
 return mid;
 }
 else{
 if (key>arr[mid]){
 low=mid+1;
 }
 else{
 high=mid-1;
 }
 binarySearchRecursive(arr,low,high,key);
 }
}
int main() {
 int n;
 scanf("%d",&n);
 int arr[n];
 for (int i=0;i<n;i++){
 scanf("%d",&arr[i]);
 }
 int key,result;
 scanf("%d",&key);
 result=binarySearchRecursive(arr,0,n-1,key);
 if(result<0){
 printf("Element not found in the array.");
 }
 else{
 printf("%d",result);
 }
 return 0;
}
```

# Analysis:

**Register No:** RTC2024BEC023

| | |
|---|---|
| Time Complexity | **- O(log n)** |
| Space Complexity | **- O(1)** |
| Logic Analysis | **- Intermediate** |
| Algorithmic Analysis | **- Intermediate** |
| Code Proficiency | **- Intermediate** |
| Coding Conventions | **- Intermediate** |
| Error Handling | **- Beginner** |
| Code Reusability | **- Beginner** |
| Code Accuracy | **- 100%** |

**Register No:** RTC2024BEC023

**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;
 scanf("%d",&size);
 int arr[size];
 int *ptr;
 ptr = arr;
 for(int i=0;i<size;i++){
 scanf("%d",&*(ptr+i));
 }
int sum=0;
for(int i=0;i<size;i++){
 sum+=*(ptr+i);
}
float average=(float)sum/(float)size;
printf("Average: %0.2f",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%** |

**Register No:** RTC2024BEC023

| Code Proficiency | **- Beginner** |
|---|---|

**Register No:** RTC2024BEC023

**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 choice;
float a,b,c;
scanf("%d",&choice);
switch(choice){
case 1:
printf("");
scanf("%f %f",&a,&b);
c= a + b;
printf(" %0.2f\n", c);
break;
case 2:
printf("");
scanf("%f %f",&a,&b);
c = a - b;
printf("%0.2f\n", c);
break;
case 3:
printf("");
scanf("%f %f",&a,&b);
c = a * b;
printf("%0.2f\n", c);
break;
case 4:
printf("");
scanf("%f %f",&a,&b);
if(b==0){
printf(" Error: Division by zero!\n");
}else{
c = a / b;
 printf(" %0.2f\n",c) ;
}
```

```
break;
case 5:
printf ("");
scanf("%f %f",&a,&b);
if(b == 0){
printf(" Error: Division by zero!\n");
break;
}else{
c = (int)a % (int)b;
printf(" %0.2f\n",c);
}
break;
case 6:
printf("");
scanf("%f",&a);
c = a * a;
printf(" %0.2f\n",c);
break;
default :
printf("Invalid operation!\n");
break;
}
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%4 == 0 && year%100 != 0)|| year%400 == 0){
 printf("Leap Year");
 }
 else{
 printf("Not a Leap Year");
 }
 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** | | |

**Register No:** RTC2024BEC023

**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 number;
 scanf("%d",&number);
 int rem,temp=0;
 int dup=number;
 int p=0;
 while (number != 0){
 p++;
 number/=10;
 }
 number=dup;
 while(number != 0){
 rem=number%10;
 temp+=pow(rem,p);
 number/=10;
 }
 if (temp==dup){
 printf("Armstrong Number\n");
 }
 else{
 printf("Not an Armstrong Number\n");
 }
 return 0;
}
```

# Analysis:

**Register No:** RTC2024BEC023

| | |
|---|---|
| Time Complexity | **- O(n)** |
| Space Complexity | **- O(1)** |
| Logic Analysis | **- Beginner** |
| Algorithmic Analysis | **- Beginner** |
| Code Proficiency | **- Beginner** |
| Coding Conventions | **- Beginner** |
| Error Handling | **- Beginner** |
| Code Reusability | **- Beginner** |
| Code Accuracy | **- 100%** |

**Register No:** RTC2024BEC023

**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>
#include <string.h>
// Function to reverse a string
void strRev(char *str, int l) {
 l = l - 1;
 char temp[50]; // Temporary array for reversed string
 int i = 0;
 while (str[i] != '\0') {
  temp[i] = str[l - i]; // Reverse string
 i++;
 }
 temp[i] = '\0'; // Null-terminate the reversed string
```

```
 printf("Reversed string: %s\n", temp);
 }
 // Function to convert string to lowercase
 void strLwr(char *str) {
 int i = 0;
 while (str[i] != '\0') {
 if (str[i] >= 'A' &&
str[i] <= 'Z') {
 str[i] +=
32; // Convert to lowercase
 }

i++;

}

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

}

// Function to convert string to uppercase

void strUpr(char *str) {

int i = 0;

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

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

str[i] -= 32; // Convert to uppercase

}

i++;

}

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

}

int main() {

char str1[100], str2[100], str3[100], str4[200] = ""; // Initialize str4 to avoid
undefined behavior

// Reading strings

scanf("%[^\n]s", str1); // Read the first string with spaces

getchar(); // To consume the newline character after the first input

scanf("%[^\n]s", str2); // Read the second string

int length = strlen(str1);

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

// Copy str1 to str3
```

```
strcpy(str3, str1);

printf("Copied string: %s\n", str3);

// Compare two strings

printf("Enter the second string for comparison: Comparing '%s' and '%s': ", str1,
str2);

int cmp = strcmp(str1, str2);

if (cmp == 0) {

printf("Both strings are equal\n");

}

if (cmp < 0) {

printf("First string is smaller\n");

}

if (cmp > 0) {

printf("First string is greater\n");

}

// Reverse the string

strcpy(str3, str1);

strRev(str3, length);

// Convert the string to lowercase

strcpy(str3, str1);

strLwr(str3);
// Convert the string to uppercase

strcpy(str3, str1);

strUpr(str3);

// Concatenate the strings

strcpy(str4, str1); // Copy str1 to str4

strcat(str4, " "); // Add a space between the strings

strcat(str4, str2); // Concatenate str2 to str4

printf("Concatenated string: %s\n", str4);

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;
 temp=a;
 a=b;
 b=temp;
}
void swapByReference(int*a,int*b){
 int temp;
 temp=*a;
 *a=*b;
 *b=temp;
}
int main() {
 int a,b;
 scanf("%d %d",&a,&b);
 swapByValue(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 | **- Beginner** |
| Space Complexity | **- O(1)** | Error Handling | **- Beginner** |
| Logic Analysis | **- Beginner** | Code Reusability | **- Beginner** |
| Algorithmic Analysis | **- Beginner** | Code Accuracy | **- 100%** |

**Register No:** RTC2024BEC023

| Code Proficiency | **- Beginner** |
|---|---|

**Register No:** RTC2024BEC023

**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 fact(int n){
 if(n==0){
 return 1;
 }
return n*fact(n-1);
}
int main() {
 int n;
 printf("");
 scanf("%d",&n);
 printf("%d = %d",n,fact(n));
 return 0;
}
```

# Analysis:

| | |
|---|---|
| Time Complexity | **- O(n)** |
| Space Complexity | **- O(n)** |
| Logic Analysis | **- Beginner** |
| Algorithmic Analysis | **- Beginner** |
| Code Proficiency | **- Beginner** |
| Coding Conventions | **- Beginner** |
| Error Handling | **- Beginner** |
| Code Reusability | **- Beginner** |
| Code Accuracy | **- 100%** |

**Register No:** RTC2024BEC023

**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 a,b;
 float c;
 scanf("%f %f",&a,&b);
 c=a * b;
 printf("%.2f",c);
 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** | | |

**Register No:** RTC2024BEC023

**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 main() {
 int n;
 scanf("%d",&n);
 int arr[n];
 for(int i=0;i<n;i++){
 scanf("%d",&arr[i]);
 }
int key,a=0;
scanf("%d",&key);
for(int i=0;i<n;i++){
 if (arr[i]==key){
 printf("%d",i);
 a+=1;
 break;
 }
}
if(a==0){
 printf("-1");
}
 return 0;
}
```

# Analysis:

**Register No:** RTC2024BEC023

| | |
|---|---|
| Time Complexity | **- O(n)** |
| Space Complexity | **- O(1)** |
| Logic Analysis | **- Beginner** |
| Algorithmic Analysis | **- Beginner** |
| Code Proficiency | **- Beginner** |
| Coding Conventions | **- Beginner** |
| Error Handling | **- Beginner** |
| Code Reusability | **- Beginner** |
| Code Accuracy | **- 100%** |

**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[49];
 float m1;
 float m2;
 float m3;
}s1;
int main() {
 char name1[51];
 scanf("%d",&s1.rollno);
 scanf("\n");
  scanf("%[^\n]",&name1);
 strcpy(s1.name,name1);
```

```
 scanf("%f",&s1.m1);
 scanf("%f",&s1.m2);
 scanf("%f",&s1.m3);
 printf("Roll No: %d\n",s1.rollno);
 printf("Name: %s\n",s1.name);
 printf("Marks in Subject 1: %0.1f\n",s1.m1);
 printf("Marks in Subject 2: %0.1f\n",s1.m2);
 printf("Marks in Subject 3: %0.1f\n",s1.m3);
 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** | | |