Programming in C

Practice Problems with Solutions — Part 5

Chapter: Arrays and Strings — For Engineering / Polytechnic Students

This is a further continuation of the Arrays and Strings practice problem set, numbered 55 onward so it can be used alongside Parts 1 to 4. It covers matrix multiplication, symmetric matrix checking, insertion sort, and further string-processing exercises, including phrase palindromes, case counting, and a first non-repeating character finder.

Section A: Arrays

Problem 55: Multiplication of Two Matrices

Problem Statement:

Write a C program to multiply two matrices, where the number of columns in the first matrix equals the number of rows in the second matrix.

Solution:

```
#include <stdio.h>
int main()
{
    int a[10][10], b[10][10], result[10][10];
    int r1, c1, r2, c2, i, j, k;

    printf("Enter rows and columns of first matrix: ");
    scanf("%d %d", &r1, &c1);
    printf("Enter elements of first matrix:\n");
    for (i = 0; i < r1; i++)
        for (j = 0; j < c1; j++)
            scanf("%d", &a[i][j]);

    printf("Enter rows and columns of second matrix: ");
    scanf("%d %d", &r2, &c2);

    if (c1 != r2)
    {
        printf("Matrix multiplication not possible.\n");
        return 0;
    }

    printf("Enter elements of second matrix:\n");
    for (i = 0; i < r2; i++)
        for (j = 0; j < c2; j++)
            scanf("%d", &b[i][j]);

    for (i = 0; i < r1; i++)
    {
        for (j = 0; j < c2; j++)
        {
            result[i][j] = 0;
```

```
                for (k = 0; k < c1; k++)
                    result[i][j] = result[i][j] + a[i][k] * b[k][j];
            }
        }

        printf("Product matrix:\n");
        for (i = 0; i < r1; i++)
        {
            for (j = 0; j < c2; j++)
                printf("%d ", result[i][j]);
            printf("\n");
        }
        return 0;
    }
```

Sample Output:

```
Product matrix:
19 22
43 50
```

Explanation:

Matrix multiplication requires the first matrix's column count to match the second matrix's row count; this is checked before proceeding. Each element of the result, result[i][j], is computed as the sum of products of the corresponding row of the first matrix and column of the second matrix, using an innermost loop over k to accumulate that sum.

Problem 56: Check Whether a Square Matrix is Symmetric

Problem Statement:

Write a C program to check whether a given square matrix is symmetric, that is, it is equal to its own transpose (a[i][j] equals a[j][i] for every i and j).

Solution:

```
#include <stdio.h>
int main()
{
    int a[10][10], n, i, j, isSymmetric = 1;

    printf("Enter size of the square matrix: ");
    scanf("%d", &n);

    printf("Enter matrix elements:\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &a[i][j]);

    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            if (a[i][j] != a[j][i])
            {
                isSymmetric = 0;
                break;
            }
        }
```

```
    }

    if (isSymmetric)
        printf("The matrix is symmetric.\n");
    else
        printf("The matrix is not symmetric.\n");

    return 0;
}
```

Sample Output:

```
Enter size of the square matrix: 3
The matrix is symmetric.
```

Explanation:

For a matrix to be symmetric, every element must equal the element in the mirrored position across the main diagonal, that is a[i][j] must equal a[j][i]. The nested loop checks this condition for every pair of positions, and a single mismatch is enough to conclude that the matrix is not symmetric.

Problem 57: Sum of Upper Triangular and Lower Triangular Elements

Problem Statement:

Write a C program to find the sum of the elements above the main diagonal (upper triangle) and the sum of the elements below the main diagonal (lower triangle) of a square matrix.

Solution:

```
#include <stdio.h>
int main()
{
    int a[10][10], n, i, j;
    int upperSum = 0, lowerSum = 0;

    printf("Enter size of the square matrix: ");
    scanf("%d", &n);

    printf("Enter matrix elements:\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &a[i][j]);

    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            if (j > i)
                upperSum = upperSum + a[i][j];
            else if (j < i)
                lowerSum = lowerSum + a[i][j];
        }
    }

    printf("Sum of upper triangle = %d\n", upperSum);
    printf("Sum of lower triangle = %d\n", lowerSum);
    return 0;
}
```

Sample Output:

```
Enter size of the square matrix: 3
Sum of upper triangle = 15
Sum of lower triangle = 21
```

Explanation:

An element belongs to the upper triangle when its column index is greater than its row index (j > i), and to the lower triangle when its column index is smaller (j < i); elements where i equals j lie on the main diagonal itself and are excluded from both sums by the if-else if structure.

Problem 58: Insertion Sort of an Array

Problem Statement:

Write a C program to sort an array of N integers in ascending order using the insertion sort technique.

Solution:

```
#include <stdio.h>
int main()
{
    int arr[50], n, i, j, key;

    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter %d elements: ", n);
    for (i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }

    printf("Sorted array: ");
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
    return 0;
}
```

Sample Output:

```
Enter number of elements: 5
Enter 5 elements: 12 11 13 5 6
Sorted array: 5 6 11 12 13
```

Explanation:

Starting from the second element, each value (key) is compared against the already-sorted portion of the array to its left. Larger elements are shifted one position to the right to make room, and key is finally inserted into its correct position, building up a fully sorted array one element at a time.

Problem 59: Count Positive, Negative, and Zero Elements in an Array

Problem Statement:

Write a C program to count the number of positive numbers, negative numbers, and zeros present in a given array.

Solution:

```
#include <stdio.h>
int main()
{
    int arr[50], n, i;
    int posCount = 0, negCount = 0, zeroCount = 0;

    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter %d elements: ", n);
    for (i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    for (i = 0; i < n; i++)
    {
        if (arr[i] > 0)
            posCount++;
        else if (arr[i] < 0)
            negCount++;
        else
            zeroCount++;
    }

    printf("Positive numbers = %d\n", posCount);
    printf("Negative numbers = %d\n", negCount);
    printf("Zeros = %d\n", zeroCount);
    return 0;
}
```

Sample Output:

```
Enter number of elements: 6
Enter 6 elements: 5 -2 0 8 0 -3
Positive numbers = 2
Negative numbers = 2
Zeros = 2
```

Explanation:

The if-else-if ladder inside the loop tests each element against three mutually exclusive categories — greater than zero, less than zero, or exactly zero — and increments the matching counter, giving a full breakdown of the array's contents by sign.

Problem 60: Rotate an Array to the Right by One Position

Problem Statement:

Write a C program to rotate the elements of an array to the right by one position, so that the last element moves to the front.

Solution:

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

```
{
    int arr[50], n, i, last;

    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter %d elements: ", n);
    for (i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    last = arr[n - 1];
    for (i = n - 1; i > 0; i--)
        arr[i] = arr[i - 1];
    arr[0] = last;

    printf("Array after right rotation: ");
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
    return 0;
}
```

Sample Output:

```
Enter number of elements: 5
Enter 5 elements: 10 20 30 40 50
Array after right rotation: 50 10 20 30 40
```

Explanation:

The last element is saved separately before it gets overwritten. The loop then runs backward, shifting every other element one position to the right, and finally the saved last value is placed at index 0, completing a one-position right rotation — the mirror image of a left rotation.

Section B: Strings

Problem 61: Check if a Phrase is a Palindrome (Ignoring Case and Spaces)

Problem Statement:

Write a C program to check whether a given phrase is a palindrome when spaces and letter case are ignored (for example, "Nurses Run").

Solution:

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str[200], clean[200];
    int i, j = 0, isPalindrome = 1;

    printf("Enter a phrase: ");
    fgets(str, sizeof(str), stdin);

    // build a cleaned version: letters only, all lowercase
    for (i = 0; str[i] != '\0'; i++)
    {
        if (str[i] >= 'A' && str[i] <= 'Z')
            clean[j++] = str[i] + 32;
```

```
        else if (str[i] >= 'a' && str[i] <= 'z')
            clean[j++] = str[i];
    }
    clean[j] = '\0';

    int len = strlen(clean);
    for (i = 0; i < len / 2; i++)
    {
        if (clean[i] != clean[len - 1 - i])
        {
            isPalindrome = 0;
            break;
        }
    }

    if (isPalindrome)
        printf("The phrase is a palindrome.\n");
    else
        printf("The phrase is not a palindrome.\n");

    return 0;
}
```

Sample Output:

Enter a phrase: Nurses Run
The phrase is a palindrome.

Explanation:

Before checking for symmetry, the program builds a clean version of the phrase that keeps only alphabet characters and converts every letter to lowercase, discarding spaces and case differences. The same two-pointer comparison used for a simple palindrome check is then applied to this cleaned string.

Problem 62: Count Uppercase and Lowercase Letters in a String

Problem Statement:

Write a C program to count the number of uppercase letters and lowercase letters present in a given string.

Solution:

```
#include <stdio.h>
int main()
{
    char str[100];
    int i, upperCount = 0, lowerCount = 0;

    printf("Enter a string: ");
    scanf("%s", str);

    for (i = 0; str[i] != '\0'; i++)
    {
        if (str[i] >= 'A' && str[i] <= 'Z')
            upperCount++;
        else if (str[i] >= 'a' && str[i] <= 'z')
            lowerCount++;
    }

    printf("Uppercase letters = %d\n", upperCount);
    printf("Lowercase letters = %d\n", lowerCount);
```

```
    return 0;
}
```

Sample Output:

```
Enter a string: PolyNotesHub
Uppercase letters = 3
Lowercase letters = 9
```

Explanation:

The loop tests each character against the uppercase range and the lowercase range in turn, incrementing the appropriate counter. Characters that are not letters at all (digits or symbols) simply fail both checks and are not counted in either total.

Problem 63: Convert an Integer to a String Without sprintf()

Problem Statement:

Write a C program to convert an integer into its equivalent string of digits, without using the standard library function sprintf().

Solution:

```
#include <stdio.h>
#include <string.h>
int main()
{
    int num, temp, i = 0, isNegative = 0;
    char digits[20], result[20];

    printf("Enter an integer: ");
    scanf("%d", &num);

    if (num < 0)
    {
        isNegative = 1;
        num = -num;
    }

    temp = num;
    if (temp == 0)
        digits[i++] = '0';
    while (temp != 0)
    {
        digits[i++] = (temp % 10) + '0';
        temp = temp / 10;
    }

    // digits are in reverse order, so build result in correct order
    int j = 0;
    if (isNegative)
        result[j++] = '-';
    while (i > 0)
        result[j++] = digits[--i];
    result[j] = '\0';

    printf("String form: %s\n", result);
    return 0;
}
```

Sample Output:

```
Enter an integer: -207
String form: -207
```

Explanation:

Digits are extracted from the number one at a time using % 10 and / 10, exactly as in a digit-reversal problem, which naturally produces them in reverse order. They are therefore stored first into digits[] and then copied into result[] back-to-front, with a leading '-' inserted first if the original number was negative.

Problem 64: Delete All Vowels from a String

Problem Statement:

Write a C program to remove all vowel characters from a given string.

Solution:

```
#include <stdio.h>
int main()
{
    char str[100], result[100];
    int i, j = 0;

    printf("Enter a string: ");
    scanf("%s", str);

    for (i = 0; str[i] != '\0'; i++)
    {
        char ch = str[i];
        if (!(ch=='a'||ch=='e'||ch=='i'||ch=='o'||ch=='u'||
              ch=='A'||ch=='E'||ch=='I'||ch=='O'||ch=='U'))
        {
            result[j] = str[i];
            j++;
        }
    }
    result[j] = '\0';

    printf("String without vowels: %s\n", result);
    return 0;
}
```

Sample Output:

```
Enter a string: Engineering
String without vowels: ngnrng
```

Explanation:

Every character of str is tested against the set of vowels. Only characters that are not vowels are copied into result, so the final string keeps every consonant and non-letter character from the original, but none of the vowels.

Problem 65: Find the First Non-Repeating Character in a String

Problem Statement:

Write a C program to find the first character in a string that does not repeat anywhere else in the string.

Solution:

```c
#include <stdio.h>
#include <string.h>
int main()
{
    char str[100];
    int freq[256] = {0};
    int i, len;

    printf("Enter a string: ");
    scanf("%s", str);
    len = strlen(str);

    for (i = 0; i < len; i++)
        freq[(int)str[i]]++;

    for (i = 0; i < len; i++)
    {
        if (freq[(int)str[i]] == 1)
        {
            printf("First non-repeating character = '%c'\n", str[i]);
            return 0;
        }
    }

    printf("No non-repeating character found.\n");
    return 0;
}
```

Sample Output:

```
Enter a string: swiss
First non-repeating character = 'w'
```

Explanation:

The first pass fills freq with the total number of times each character appears anywhere in the string. The second pass then walks through the string again in its original order and returns the first character whose overall frequency is exactly 1, guaranteeing it is genuinely the earliest non-repeating character.

Problem 66: Check if Two Strings are Equal Ignoring Case

Problem Statement:

Write a C program to check whether two strings are equal when letter case is ignored (a case-insensitive comparison).

Solution:

```c
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[100], str2[100];
    int i, areEqual = 1;

    printf("Enter first string: ");
    scanf("%s", str1);
    printf("Enter second string: ");
    scanf("%s", str2);
```

```
    if (strlen(str1) != strlen(str2))
        areEqual = 0;
    else
    {
        for (i = 0; str1[i] != '\0'; i++)
        {
            char c1 = str1[i], c2 = str2[i];
            if (c1 >= 'A' && c1 <= 'Z') c1 = c1 + 32;
            if (c2 >= 'A' && c2 <= 'Z') c2 = c2 + 32;
            if (c1 != c2)
            {
                areEqual = 0;
                break;
            }
        }
    }

    if (areEqual)
        printf("The strings are equal (ignoring case).\n");
    else
        printf("The strings are not equal.\n");

    return 0;
}
```

Sample Output:

Enter first string: PolyNotes
Enter second string: polynotes
The strings are equal (ignoring case).

Explanation:

Before comparing each pair of characters, both are converted to lowercase using local copies c1 and c2, so a character's case no longer affects the comparison. Two strings are judged equal only if they have the same length and every corresponding pair of characters matches after this case normalization.