Programming in C

Practice Problems with Solutions — Part 4

Chapter: Arrays and Strings — For Engineering / Polytechnic Students

This is a further continuation of the Arrays and Strings practice problem set, numbered 43 onward so it can be used alongside Parts 1 to 3. It covers array reversal, selection sort, 2D matrix extremes, and further string-processing exercises, including character frequency, ASCII values, and string rotation.

Section A: Arrays

Problem 43: Sum of Elements at Even and Odd Index Positions

Problem Statement:

Write a C program to separately find the sum of array elements located at even index positions and the sum of array elements located at odd index positions.

Solution:

```
#include <stdio.h>
int main()
{
    int arr[50], n, i, evenIndexSum = 0, oddIndexSum = 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 (i % 2 == 0)
            evenIndexSum = evenIndexSum + arr[i];
        else
            oddIndexSum = oddIndexSum + arr[i];
    }

    printf("Sum at even index positions = %d\n", evenIndexSum);
    printf("Sum at odd index positions = %d\n", oddIndexSum);
    return 0;
}
```

Sample Output:

```
Enter number of elements: 6
Enter 6 elements: 10 20 30 40 50 60
Sum at even index positions = 90
Sum at odd index positions = 120
```

Explanation:

Here the position (index) of each element decides where it is added, not its value. Index 0, 2, 4 (even positions) contribute to evenIndexSum, while index 1, 3, 5 (odd positions) contribute to oddIndexSum, which is different from separating elements by whether their values are even or odd.

Problem 44: Reverse an Array in Place

Problem Statement:

Write a C program to reverse the elements of an array in place, without using a second array.

Solution:

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

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

    start = 0;
    end = n - 1;
    while (start < end)
    {
        temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;
        start++;
        end--;
    }

    printf("Reversed 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: 10 20 30 40 50
Reversed array: 50 40 30 20 10
```

Explanation:

Two index pointers, start and end, move toward each other from opposite ends of the array. On each pass, the elements they point to are swapped, and the pointers step inward; the loop stops once start meets or crosses end, by which point the whole array has been reversed in place.

Problem 45: Largest and Smallest Element in a 2D Matrix

Problem Statement:

Write a C program to find the largest and smallest elements present anywhere in a two dimensional (M x N) matrix.

Solution:

```
#include <stdio.h>
int main()
{
    int a[10][10], rows, cols, i, j;

    printf("Enter rows and columns: ");
    scanf("%d %d", &rows, &cols);

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

    int largest = a[0][0], smallest = a[0][0];
    for (i = 0; i < rows; i++)
    {
        for (j = 0; j < cols; j++)
        {
            if (a[i][j] > largest)
                largest = a[i][j];
            if (a[i][j] < smallest)
                smallest = a[i][j];
        }
    }

    printf("Largest = %d\n", largest);
    printf("Smallest = %d\n", smallest);
    return 0;
}
```

Sample Output:

```
Enter rows and columns: 2 3
Enter matrix elements:
4 9 2 7 1 8
Largest = 9
Smallest = 1
```

Explanation:

largest and smallest are both initialized with the first element of the matrix, a[0][0]. The nested loop then visits every element in turn, comparing it against the current largest and smallest and updating them whenever a new extreme value is found, exactly as with a one dimensional array, but scanning both rows and columns.

Problem 46: Selection Sort of an Array

Problem Statement:

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

Solution:

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

    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 - 1; i++)
    {
        minIndex = i;
        for (j = i + 1; j < n; j++)
        {
            if (arr[j] < arr[minIndex])
                minIndex = j;
        }
        if (minIndex != i)
        {
            temp = arr[i];
            arr[i] = arr[minIndex];
            arr[minIndex] = temp;
        }
    }

    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: 29 10 14 37 3
Sorted array: 3 10 14 29 37
```

Explanation:

For every position i, the inner loop scans the remaining unsorted part of the array to find the index of the smallest element, minIndex. That smallest element is then swapped into position i, so with each pass of the outer loop, one more element settles into its correct sorted position.

Problem 47: Check Whether Two Arrays are Equal

Problem Statement:

Write a C program to check whether two arrays of the same size are equal, that is, they contain the same elements in the same order.

Solution:

```
#include <stdio.h>
int main()
{
    int a[50], b[50], n, i, isEqual = 1;

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

    printf("Enter elements of first array: ");
    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);
```

```
    printf("Enter elements of second array: ");
    for (i = 0; i < n; i++)
        scanf("%d", &b[i]);

    for (i = 0; i < n; i++)
    {
        if (a[i] != b[i])
        {
            isEqual = 0;
            break;
        }
    }

    if (isEqual)
        printf("The arrays are equal.\n");
    else
        printf("The arrays are not equal.\n");

    return 0;
}
```

Sample Output:

```
Enter size of the arrays: 4
Enter elements of first array: 1 2 3 4
Enter elements of second array: 1 2 3 4
The arrays are equal.
```

Explanation:

The loop compares the two arrays index by index. If any pair of corresponding elements differs, the arrays cannot be equal, so isEqual is set to 0 and break exits the comparison immediately, since one mismatch is enough to decide the answer.

Problem 48: Average of Elements in a Two Dimensional Array

Problem Statement:

Write a C program to find the average of all the elements stored in a two dimensional (M x N) array.

Solution:

```
#include <stdio.h>
int main()
{
    int a[10][10], rows, cols, i, j, sum = 0;

    printf("Enter rows and columns: ");
    scanf("%d %d", &rows, &cols);

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

    printf("Average = %.2f\n", (float)sum / (rows * cols));
```

```
    return 0;
}
```

Sample Output:

```
Enter rows and columns: 2 2
Enter matrix elements:
10 20 30 40
Average = 25.00
```

Explanation:

As every element of the matrix is read, it is added to sum. Once the entire matrix has been scanned, dividing sum by the total number of elements (rows multiplied by columns) gives the average, with the sum cast to float first to avoid integer division truncation.

Section B: Strings

Problem 49: Frequency of Each Character in a String

Problem Statement:

Write a C program to find and display the frequency of each distinct alphabet character present in a given string.

Solution:

```
#include <stdio.h>
int main()
{
    char str[100];
    int freq[26] = {0};
    int i;

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

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

    for (i = 0; i < 26; i++)
    {
        if (freq[i] > 0)
            printf("%c occurs %d time(s)\n", 'a' + i, freq[i]);
    }
    return 0;
}
```

Sample Output:

```
Enter a string: programming
g occurs 2 time(s)
i occurs 1 time(s)
m occurs 2 time(s)
n occurs 1 time(s)
```

```
o occurs 1 time(s)
p occurs 1 time(s)
r occurs 2 time(s)
```

Explanation:

The freq array has 26 slots, one for every letter of the alphabet. Subtracting 'a' (or 'A') from a character gives a number from 0 to 25 that identifies its matching slot, so freq[str[i] - 'a']++ increments the count for exactly that letter regardless of where it appears in the string.

Problem 50: Convert a String of Digits into an Integer

Problem Statement:

Write a C program to convert a string that represents a number into its equivalent integer value, without using the standard library function atoi().

Solution:

```
#include <stdio.h>
int main()
{
    char str[20];
    int i, number = 0;

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

    for (i = 0; str[i] != '\0'; i++)
        number = number * 10 + (str[i] - '0');

    printf("Integer value = %d\n", number);
    return 0;
}
```

Sample Output:

```
Enter a numeric string: 4582
Integer value = 4582
```

Explanation:

For each digit character, subtracting the character '0' gives its actual numeric value (since digit characters are consecutive in ASCII order). number is rebuilt on every pass by shifting its existing digits one place left (multiplying by 10) and adding in the new digit, exactly reconstructing the original number from its string form.

Problem 51: Find the Position of the First Occurrence of a Character

Problem Statement:

Write a C program to find the position (index) of the first occurrence of a given character in a string.

Solution:

```
#include <stdio.h>
int main()
{
    char str[100], ch;
    int i, position = -1;

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

```
    printf("Enter character to find: ");
    scanf(" %c", &ch);

    for (i = 0; str[i] != '\0'; i++)
    {
        if (str[i] == ch)
        {
            position = i + 1;
            break;
        }
    }

    if (position != -1)
        printf("'%c' first found at position %d\n", ch, position);
    else
        printf("'%c' not found in the string.\n", ch);

    return 0;
}
```

Sample Output:

```
Enter a string: polynoteshub
Enter character to find: t
't' first found at position 6
```

Explanation:

The loop scans the string from the beginning and stops with break as soon as it finds a character matching ch, recording its 1-based position. If the loop reaches the end of the string without a match, position remains -1, signalling that the character was not found.

Problem 52: Check Whether a String is a Rotation of Another String

Problem Statement:

Write a C program to check whether one string is a rotation of another string of the same length (for example, "waterbottle" and "erbottlewat" are rotations of each other).

Solution:

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[100], str2[100], combined[200];
    int i, j, len1, len2, match;

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

    len1 = strlen(str1);
    len2 = strlen(str2);

    if (len1 != len2)
    {
        printf("Not a rotation (different lengths).\n");
        return 0;
```

```
    }

    // combined = str1 followed by str1 again, e.g. "abcabc"
    strcpy(combined, str1);
    strcat(combined, str1);

    int found = 0;
    for (i = 0; i <= (int)strlen(combined) - len2; i++)
    {
        match = 1;
        for (j = 0; j < len2; j++)
        {
            if (combined[i + j] != str2[j])
            {
                match = 0;
                break;
            }
        }
        if (match)
        {
            found = 1;
            break;
        }
    }

    if (found)
        printf("'%s' is a rotation of '%s'.\n", str2, str1);
    else
        printf("'%s' is not a rotation of '%s'.\n", str2, str1);

    return 0;
}
```

Sample Output:

```
Enter first string: abcde
Enter second string: cdeab
'cdeab' is a rotation of 'abcde'.
```

Explanation:

A useful trick for this problem is that every possible rotation of str1 always appears somewhere inside str1 concatenated with itself (for example, all rotations of "abc" appear inside "abcabc"). So the program builds this doubled string and then searches it for str2 using the same substring-matching logic as a manual strstr().

Problem 53: Toggle the Case of Each Alphabet in a String

Problem Statement:

Write a C program to toggle the case of every alphabet character in a string — converting each uppercase letter to lowercase and each lowercase letter to uppercase.

Solution:

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

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

    for (i = 0; str[i] != '\0'; i++)
    {
        if (str[i] >= 'a' && str[i] <= 'z')
            str[i] = str[i] - 32;
        else if (str[i] >= 'A' && str[i] <= 'Z')
            str[i] = str[i] + 32;
    }

    printf("Toggled string: %s\n", str);
    return 0;
}
```

Sample Output:

```
Enter a string: PolyNotesHub
Toggled string: pOLYnOTEShUB
```

Explanation:

Every character is checked against both letter ranges. A lowercase letter has 32 subtracted from its ASCII value to become uppercase, while an uppercase letter has 32 added to become lowercase, so each alphabet character in the string ends up in the opposite case from where it started.

Problem 54: Print Each Character of a String with its ASCII Value

Problem Statement:

Write a C program to print every character of a given string along with its corresponding ASCII value.

Solution:

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

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

    for (i = 0; str[i] != '\0'; i++)
        printf("'%c' -> %d\n", str[i], str[i]);

    return 0;
}
```

Sample Output:

```
Enter a string: CAB
'C' -> 67
'A' -> 65
'B' -> 66
```

Explanation:

Since a char in C is really stored as a small integer, printing str[i] with the %d format specifier displays that character's underlying ASCII value directly, alongside its usual printed form obtained with %c.