Programming in C

Practice Problems with Solutions — Part 2

Chapter: Arrays and Strings — For Engineering / Polytechnic Students

This is a continuation of the Arrays and Strings practice problem set, numbered 19 onward so it can be used alongside Part 1. It covers further array operations such as merging, rotation, and frequency counting, along with additional string-processing exercises.

Section A: Arrays

Problem 19: Merge Two Arrays into a Third Array

Problem Statement:

Write a C program to merge two arrays into a single third array by placing all elements of the first array followed by all elements of the second array.

Solution:

```
#include <stdio.h>
int main()
{
    int a[30], b[30], merged[60];
    int n1, n2, i;

    printf("Enter size and elements of first array: ");
    scanf("%d", &n1);
    for (i = 0; i < n1; i++)
        scanf("%d", &a[i]);

    printf("Enter size and elements of second array: ");
    scanf("%d", &n2);
    for (i = 0; i < n2; i++)
        scanf("%d", &b[i]);

    for (i = 0; i < n1; i++)
        merged[i] = a[i];
    for (i = 0; i < n2; i++)
        merged[n1 + i] = b[i];

    printf("Merged array: ");
    for (i = 0; i < n1 + n2; i++)
        printf("%d ", merged[i]);
    printf("\n");
    return 0;
}
```

Sample Output:

```
Merged array: 1 3 5 2 4 6
```

Explanation:

The first loop copies all n1 elements of a directly into the start of merged. The second loop copies the n2 elements of b into merged as well, but offsets each index by n1 so that they are placed immediately after the elements already copied from a.

Problem 20: Find the Missing Number in an Array of 1 to N

Problem Statement:

An array contains N-1 distinct numbers taken from the range 1 to N, with exactly one number missing. Write a C program to find the missing number.

Solution:

```
#include <stdio.h>
int main()
{
    int arr[50], n, i;
    long expectedSum, actualSum = 0;

    printf("Enter N (the range is 1 to N): ");
    scanf("%d", &n);

    printf("Enter %d numbers (one number from 1 to %d is missing): ", n - 1, n);
    for (i = 0; i < n - 1; i++)
    {
        scanf("%d", &arr[i]);
        actualSum = actualSum + arr[i];
    }

    expectedSum = (long)n * (n + 1) / 2;
    printf("Missing number = %ld\n", expectedSum - actualSum);

    return 0;
}
```

Sample Output:

```
Enter N (the range is 1 to N): 5
Enter 4 numbers (one number from 1 to 5 is missing): 1 2 4 5
Missing number = 3
```

Explanation:

The sum of all numbers from 1 to N has a well-known formula, N(N+1)/2. Subtracting the actual sum of the given array from this expected sum directly yields the one value that is missing, without needing to search or sort the array.

Problem 21: Rotate an Array to the Left by One Position

Problem Statement:

Write a C program to rotate the elements of an array to the left by one position, so that the first element moves to the end.

Solution:

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

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

    first = arr[0];
    for (i = 0; i < n - 1; i++)
        arr[i] = arr[i + 1];
    arr[n - 1] = first;

    printf("Array after left 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 left rotation: 20 30 40 50 10
```

Explanation:

The original first element is saved separately before it gets overwritten. Every other element is then shifted one position to the left, and finally the saved first value is placed at the last index, completing a one-position left rotation.

Problem 22: Check Whether an Array is Sorted in Ascending Order

Problem Statement:

Write a C program to check whether the elements of an array are arranged in ascending order.

Solution:

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

    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++)
    {
        if (arr[i] > arr[i + 1])
        {
            isSorted = 0;
            break;
        }
    }

    if (isSorted)
        printf("The array is sorted in ascending order.\n");
    else
```

```
            printf("The array is not sorted.\n");

        return 0;
    }
```

Sample Output:

```
Enter number of elements: 5
Enter 5 elements: 2 4 4 9 15
The array is sorted in ascending order.
```

Explanation:

The loop compares every element with the one immediately after it. If any element is found to be greater than its successor, the array cannot be in ascending order, so isSorted is set to 0 and break exits the loop right away.

Problem 23: Sum of Positive and Negative Numbers in an Array

Problem Statement:

Write a C program to separately calculate the sum of all positive numbers and the sum of all negative numbers stored in an array.

Solution:

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

    printf("Sum of positive numbers = %d\n", posSum);
    printf("Sum of negative numbers = %d\n", negSum);
    return 0;
}
```

Sample Output:

```
Enter number of elements: 6
Enter 6 elements: 5 -3 8 -7 2 -1
Sum of positive numbers = 15
Sum of negative numbers = -11
```

Explanation:

The if-else inside the loop tests the sign of each element and routes it into the appropriate running total, so posSum accumulates only non-negative values and negSum accumulates only negative ones.

Problem 24: Frequency of Each Element in an Array

Problem Statement:

Write a C program to find and display the frequency (number of occurrences) of each distinct element in an array.

Solution:

```
#include <stdio.h>
int main()
{
    int arr[50], visited[50] = {0}, n, i, j, count;

    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 (visited[i] == 1)
            continue;   // this value's frequency was already printed

        count = 1;
        for (j = i + 1; j < n; j++)
        {
            if (arr[i] == arr[j])
            {
                visited[j] = 1;
                count++;
            }
        }
        printf("%d occurs %d time(s)\n", arr[i], count);
    }
    return 0;
}
```

Sample Output:

```
Enter number of elements: 6
Enter 6 elements: 4 2 4 5 2 4
4 occurs 3 time(s)
2 occurs 2 time(s)
5 occurs 1 time(s)
```

Explanation:

The visited array marks positions whose value has already been counted and printed, so continue is used to skip them. For every new (unvisited) value, the inner loop scans the rest of the array, marking and counting every further occurrence of the same value.

Section B: Strings

Problem 25: Find the Largest Word in a Sentence

Problem Statement:

Write a C program to find and print the longest word in a given sentence.

Solution:

```
#include <stdio.h>
#include <string.h>
int main()
{
    char sentence[200], word[50], largest[50];
    int i = 0, j = 0;
    int maxLen = 0;

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

    while (1)
    {
        if (sentence[i] != ' ' && sentence[i] != '\n' && sentence[i] != '\0')
        {
            word[j] = sentence[i];
            j++;
        }
        else
        {
            word[j] = '\0';
            if (strlen(word) > maxLen)
            {
                maxLen = strlen(word);
                strcpy(largest, word);
            }
            j = 0;
            if (sentence[i] == '\0')
                break;
        }
        i++;
    }

    printf("Largest word = %s (length %d)\n", largest, maxLen);
    return 0;
}
```

Sample Output:

```
Enter a sentence: C programming builds strong fundamentals
Largest word = fundamentals (length 13)
```

Explanation:

The sentence is scanned character by character to build up one word at a time in word. Whenever a word boundary (space, newline, or the end of the string) is reached, the just-completed word's length is compared against maxLen, and largest is updated using strcpy() whenever a longer word is found.

Problem 26: Convert a Sentence to Title Case

Problem Statement:

Write a C program to convert a sentence to title case, where the first letter of every word is capitalised and the rest remain unchanged.

Solution:

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

```
{
    char str[200];
    int i;

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

    if (str[0] >= 'a' && str[0] <= 'z')
        str[0] = str[0] - 32;

    for (i = 1; str[i] != '\0'; i++)
    {
        if (str[i - 1] == ' ' && str[i] >= 'a' && str[i] <= 'z')
            str[i] = str[i] - 32;
    }

    printf("Title case: %s", str);
    return 0;
}
```

Sample Output:

```
Enter a sentence: web by arun
Title case: Web By Arun
```

Explanation:

The very first character is capitalised as a special case. After that, the loop capitalises any lowercase letter that immediately follows a space, since a space marks the start of a new word, while every other character is left untouched.

Problem 27: Remove Duplicate Characters from a String

Problem Statement:

Write a C program to remove duplicate characters from a string, keeping only the first occurrence of each character.

Solution:

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

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

    for (i = 0; str[i] != '\0'; i++)
    {
        isDuplicate = 0;
        for (j = 0; j < resLen; j++)
        {
            if (result[j] == str[i])
            {
                isDuplicate = 1;
                break;
            }
```

```
            }
            if (!isDuplicate)
            {
                result[resLen] = str[i];
                resLen++;
            }
        }
        result[resLen] = '\0';

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

Sample Output:

```
Enter a string: programming
String after removing duplicates: progamin
```

Explanation:

For every character of str, the inner loop checks whether that character already exists in result. It is appended to result only if it has not appeared before, so each distinct character is kept exactly once, in its original order of first appearance.

Problem 28: Check Whether a String Starts With a Given Prefix

Problem Statement:

Write a C program to check whether a given string starts with a specified prefix, without using the library function strstr().

Solution:

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str[100], prefix[50];
    int i, isPrefix = 1;

    printf("Enter the main string: ");
    scanf("%s", str);
    printf("Enter the prefix to check: ");
    scanf("%s", prefix);

    for (i = 0; prefix[i] != '\0'; i++)
    {
        if (str[i] != prefix[i])
        {
            isPrefix = 0;
            break;
        }
    }

    if (isPrefix)
        printf("\"%s\" starts with \"%s\".\n", str, prefix);
    else
        printf("\"%s\" does not start with \"%s\".\n", str, prefix);

    return 0;
```

```
}
```

Sample Output:

```
Enter the main string: polynoteshub
Enter the prefix to check: poly
"polynoteshub" starts with "poly".
```

Explanation:

The loop runs only as far as the length of prefix, comparing each of its characters against the character at the same position in str. If every character matches, str genuinely begins with prefix; a single mismatch is enough to disqualify it, so break exits immediately.

Problem 29: Replace All Occurrences of a Character in a String

Problem Statement:

Write a C program to replace every occurrence of a specified character in a string with another given character.

Solution:

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

    printf("Enter a string: ");
    scanf("%s", str);
    printf("Enter character to replace and its replacement: ");
    scanf(" %c %c", &oldChar, &newChar);

    for (i = 0; str[i] != '\0'; i++)
    {
        if (str[i] == oldChar)
            str[i] = newChar;
    }

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

Sample Output:

```
Enter a string: programming
Enter character to replace and its replacement: g G
Updated string: proGramminG
```

Explanation:

The loop scans every character of the string up to the null terminator, and whenever the current character matches oldChar, it is overwritten in place with newChar. Since replacement does not change the string's length, no extra array or resizing is needed.

Problem 30: Compare Two Strings Without Using strcmp()

Problem Statement:

Write a C program to compare two strings for equality character by character, without using the standard library function strcmp().

Solution:

```
#include <stdio.h>
int main()
{
    char str1[100], str2[100];
    int i = 0, areEqual = 1;

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

    while (str1[i] != '\0' && str2[i] != '\0')
    {
        if (str1[i] != str2[i])
        {
            areEqual = 0;
            break;
        }
        i++;
    }

    // if one string is a prefix of the other but shorter, they are still unequal
    if (str1[i] != str2[i])
        areEqual = 0;

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

    return 0;
}
```

Sample Output:

```
Enter first string: hello
Enter second string: hello
The strings are equal.
```

Explanation:

The while loop compares corresponding characters of both strings until it reaches the end of either one, stopping early with areEqual set to 0 on the first mismatch. The final check after the loop catches the case where one string is simply shorter than the other, since their characters at index i would then differ (one being '\0' and the other not).