Programming in C

Practice Problems with Solutions

Chapter: Arrays and Strings — For Engineering / Polytechnic Students

This booklet contains solved programming problems covering one dimensional and two dimensional arrays, character arrays, and string handling. Each problem includes the problem statement, a complete working C program, sample output, and a short explanation of the logic used. Students are encouraged to first attempt each problem on their own and then compare their approach with the given solution.

Section A: Arrays

Problem 1: Largest and Smallest Element in an Array

Problem Statement:

Write a C program to find the largest and smallest elements in a one dimensional array of N integers.

Solution:

```
#include <stdio.h>
int main()
{
    int n, i, arr[50];
    printf("Enter number of elements: ");
    scanf("%d", &n);

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

    int largest = arr[0], smallest = arr[0];
    for (i = 1; i < n; i++)
    {
        if (arr[i] > largest)
            largest = arr[i];
        if (arr[i] < smallest)
            smallest = arr[i];
    }

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

Sample Output:

```
Enter number of elements: 5
Enter 5 elements: 12 45 3 67 21
Largest = 67
Smallest = 3
```

Explanation:

Both largest and smallest are initialized with the first element of the array. The loop then compares every remaining element against the current largest and smallest, updating them whenever a new extreme value is found.

Problem 2: Sum and Average of Array Elements

Problem Statement:

Write a C program to calculate the sum and average of the elements stored in a one dimensional array.

Solution:

```
#include <stdio.h>
int main()
{
    int n, i, arr[50], sum = 0;
    printf("Enter number of elements: ");
    scanf("%d", &n);

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

    printf("Sum = %d\n", sum);
    printf("Average = %.2f\n", (float)sum / n);
    return 0;
}
```

Sample Output:

```
Enter number of elements: 4
Enter 4 elements: 10 20 30 40
Sum = 100
Average = 25.00
```

Explanation:

Each array element is added to sum as soon as it is read, using array indexing arr[i] inside the for loop. The average is obtained by casting the sum to float before dividing, to avoid integer division truncation.

Problem 3: Addition of Two Matrices (2D Array)

Problem Statement:

Write a C program to add two 2x2 matrices using two dimensional arrays.

Solution:

```
#include <stdio.h>
int main()
{
    int a[2][2], b[2][2], sum[2][2], i, j;

    printf("Enter elements of first matrix (2x2): ");
    for (i = 0; i < 2; i++)
        for (j = 0; j < 2; j++)
            scanf("%d", &a[i][j]);

    printf("Enter elements of second matrix (2x2): ");
```

```
    for (i = 0; i < 2; i++)
        for (j = 0; j < 2; j++)
            scanf("%d", &b[i][j]);

    for (i = 0; i < 2; i++)
        for (j = 0; j < 2; j++)
            sum[i][j] = a[i][j] + b[i][j];

    printf("Sum of the matrices:\n");
    for (i = 0; i < 2; i++)
    {
        for (j = 0; j < 2; j++)
            printf("%d ", sum[i][j]);
        printf("\n");
    }
    return 0;
}
```

Sample Output:

```
Sum of the matrices:
6 8
10 12
```

Explanation:

Three nested-loop pairs are used: two to read the two matrices, and one to add them element by element, matching row and column indices, and store the result in the sum matrix.

Problem 4: Linear Search in an Array

Problem Statement:

Write a C program to search for a given element in an array using linear search, and report its position if found.

Solution:

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

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

    printf("Enter element to search: ");
    scanf("%d", &key);

    for (i = 0; i < n; i++)
    {
        if (arr[i] == key)
        {
            printf("Element found at position %d\n", i + 1);
            found = 1;
            break;
        }
    }
```

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

    return 0;
}
```

Sample Output:

```
Enter number of elements: 5
Enter 5 elements: 4 9 15 22 30
Enter element to search: 15
Element found at position 3
```

Explanation:

The loop compares the key with each array element in turn. As soon as a match is found, its 1-based position is printed and break exits the loop early, avoiding unnecessary comparisons.

Problem 5: Bubble Sort an Array

Problem Statement:

Write a C program to sort an array of N integers in ascending order using the bubble sort technique with nested loops.

Solution:

```
#include <stdio.h>
int main()
{
    int arr[50], n, i, j, 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++)
    {
        for (j = 0; j < n - 1 - i; j++)
        {
            if (arr[j] > arr[j + 1])
            {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = 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: 40 10 30 5 25
```

```
Sorted array: 5 10 25 30 40
```

Explanation:

The outer loop controls how many passes are made, and the inner loop compares each pair of adjacent elements, swapping them if they are out of order. After each full pass, the next largest unsorted element 'bubbles up' into its correct position.

Problem 6: Second Largest Element in an Array

Problem Statement:

Write a C program to find the second largest element in an array of N integers.

Solution:

```
#include <stdio.h>
int main()
{
    int arr[50], n, i;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter %d elements: ", n);
    for (i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    int first = arr[0], second = -2147483648;
    for (i = 1; i < n; i++)
    {
        if (arr[i] > first)
        {
            second = first;
            first = arr[i];
        }
        else if (arr[i] > second && arr[i] != first)
        {
            second = arr[i];
        }
    }

    printf("Largest = %d\n", first);
    printf("Second Largest = %d\n", second);
    return 0;
}
```

Sample Output:

```
Enter number of elements: 5
Enter 5 elements: 12 45 3 45 21
Largest = 45
Second Largest = 21
```

Explanation:

first and second track the largest and second largest values seen so far. Whenever a new element beats first, the old first becomes the new second before first is updated; otherwise, if the element beats second (and is not a repeat of first), second alone is updated.

Problem 7: Transpose of a Matrix

Problem Statement:

Write a C program to find the transpose of a given M x N matrix using a two dimensional array.

Solution:

```
#include <stdio.h>
int main()
{
    int a[10][10], t[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]);

    for (i = 0; i < rows; i++)
        for (j = 0; j < cols; j++)
            t[j][i] = a[i][j];

    printf("Transpose of the matrix:\n");
    for (i = 0; i < cols; i++)
    {
        for (j = 0; j < rows; j++)
            printf("%d ", t[i][j]);
        printf("\n");
    }
    return 0;
}
```

Sample Output:

```
Transpose of the matrix:
1 4
2 5
3 6
```

Explanation:

Taking the transpose means the value at row i, column j of the original matrix moves to row j, column i of the new matrix. This is achieved directly with t[j][i] = a[i][j] inside the nested loop, and the transposed matrix ends up with rows and columns swapped.

Problem 8: Insert an Element at a Given Position

Problem Statement:

Write a C program to insert a new element into an array at a specified position, shifting the remaining elements to make room.

Solution:

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

    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter %d elements: ", n);
```

```
    for (i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    printf("Enter position (1 to %d) and value to insert: ", n + 1);
    scanf("%d %d", &pos, &value);

    for (i = n; i >= pos; i--)
        arr[i] = arr[i - 1];
    arr[pos - 1] = value;
    n++;

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

Sample Output:

```
Enter number of elements: 4
Enter 4 elements: 10 20 30 40
Enter position (1 to 5) and value to insert: 2 15
Array after insertion: 10 15 20 30 40
```

Explanation:

The loop runs backward from the last index down to the insertion position, shifting every element one place to the right so that a gap opens up at the desired position, after which the new value is placed directly into that gap.

Problem 9: Delete an Element at a Given Position

Problem Statement:

Write a C program to delete the element at a specified position from an array, shifting the remaining elements to close the gap.

Solution:

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

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

    printf("Enter position to delete (1 to %d): ", n);
    scanf("%d", &pos);

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

    printf("Array after deletion: ");
    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
Enter position to delete (1 to 5): 3
Array after deletion: 10 20 40 50
```

Explanation:

Starting from the position to be deleted, every following element is shifted one place to the left, overwriting the deleted value, and the array's logical size is reduced by one to reflect the removal.

Section B: Strings

Problem 10: Count Vowels and Consonants in a String

Problem Statement:

Write a C program to count the number of vowels and consonants present in a given string.

Solution:

```
#include <stdio.h>
int main()
{
    char str[100];
    int i, vowels = 0, consonants = 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')
            vowels++;
        else if ((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z'))
            consonants++;
    }

    printf("Vowels = %d\n", vowels);
    printf("Consonants = %d\n", consonants);
    return 0;
}
```

Sample Output:

```
Enter a string: Engineering
Vowels = 5
Consonants = 6
```

Explanation:

The loop walks through the character array until it reaches the null terminator ('\0'), checking each character against the set of vowels and counting all remaining alphabetic characters as consonants.

Problem 11: Reverse a String Without Using a Library Function

Problem Statement:

Write a C program to reverse a given string without using the standard library function strrev().

Solution:

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

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

    len = strlen(str);
    for (i = len - 1, j = 0; i >= 0; i--, j++)
        rev[j] = str[i];
    rev[j] = '\0';

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

Sample Output:

```
Enter a string: Polytechnic
Reversed string: cinhcetyloP
```

Explanation:

strlen() is used only to find the length of the string; the actual reversal copies characters from the last index of str down to the first, into rev starting from index 0, and finally places a null terminator to close the new string.

Problem 12: Check Whether a String is a Palindrome

Problem Statement:

Write a C program to check whether a given string reads the same forwards and backwards (a palindrome).

Solution:

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

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

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

```
    }

    if (isPalindrome)
        printf("%s is a palindrome.\n", str);
    else
        printf("%s is not a palindrome.\n", str);

    return 0;
}
```

Sample Output:

```
Enter a string: madam
madam is a palindrome.
```

Explanation:

The loop compares characters from the two ends of the string moving inward (index i against index len-1-i). If any pair does not match, isPalindrome is set to 0 and break exits immediately, since one mismatch is enough to disqualify the string.

Problem 13: Substring Extraction, Concatenation, and Comparison

Problem Statement:

Write a C program that stores a string, extracts the leftmost 3 characters, the rightmost 3 characters, and the middle 3 characters, then concatenates two separate strings and compares them using strcmp().

Solution:

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str[] = "Programming";
    char left[10], right[10], middle[10];
    int len = strlen(str);

    strncpy(left, str, 3);
    left[3] = '\0';

    strncpy(right, str + (len - 3), 3);
    right[3] = '\0';

    strncpy(middle, str + 4, 3);
    middle[3] = '\0';

    printf("Left 3    : %s\n", left);
    printf("Right 3   : %s\n", right);
    printf("Middle 3  : %s\n", middle);

    char first[20] = "Web";
    char second[] = "ByArun";
    strcat(first, second);
    printf("Concatenated: %s\n", first);

    if (strcmp(left, right) == 0)
        printf("left and right are the same.\n");
    else
        printf("left and right are different.\n");
```

```
    return 0;
}
```

Sample Output:

```
Left 3    : Pro
Right 3   : ing
Middle 3  : ram
Concatenated: WebByArun
Left and right are different.
```

Explanation:

strncpy() copies a fixed number of characters starting from a chosen offset (0 for left, len-3 for right, 4 for middle), and a null terminator is placed manually after each copy. strcat() then joins two separate strings, and strcmp() compares two strings and returns zero only when they are identical.

Problem 14: Count Occurrences of a Character in a String

Problem Statement:

Write a C program to count how many times a given character appears in a string.

Solution:

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

    printf("Enter a string: ");
    scanf("%s", str);
    printf("Enter character to count: ");
    scanf(" %c", &ch);

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

    printf("'%c' occurs %d time(s) in the string.\n", ch, count);
    return 0;
}
```

Sample Output:

```
Enter a string: programming
Enter character to count: r
'r' occurs 2 time(s) in the string.
```

Explanation:

The for loop scans the character array from index 0 until the null terminator, incrementing count every time the current character matches ch. A leading space in " %c" is used in scanf to skip any leftover newline from the previous input.

Problem 15: Check if Two Strings are Anagrams

Problem Statement:

Write a C program to check whether two given strings are anagrams of each other, that is, whether they contain exactly the same characters with the same frequency, in any order.

Solution:

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[100], str2[100];
    int count[256] = {0};
    int i, isAnagram = 1;

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

    if (strlen(str1) != strlen(str2))
        isAnagram = 0;
    else
    {
        for (i = 0; str1[i] != '\0'; i++)
            count[(int)str1[i]]++;
        for (i = 0; str2[i] != '\0'; i++)
            count[(int)str2[i]]--;
        for (i = 0; i < 256; i++)
        {
            if (count[i] != 0)
            {
                isAnagram = 0;
                break;
            }
        }
    }

    if (isAnagram)
        printf("The strings are anagrams.\n");
    else
        printf("The strings are not anagrams.\n");

    return 0;
}
```

Sample Output:

```
Enter first string: listen
Enter second string: silent
The strings are anagrams.
```

Explanation:

The count array tallies how often each possible character appears: it is incremented for every character of str1 and decremented for every character of str2. If the two strings are true anagrams, every count returns to exactly zero; any non-zero value means the character frequencies did not match.

Problem 16: Convert a String to Uppercase Without a Library Function

Problem Statement:

Write a C program to convert a lowercase string to uppercase without using the standard library function toupper().

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;
    }

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

Sample Output:

```
Enter a string: polynoteshub
Uppercase string: POLYNOTESHUB
```

Explanation:

In the ASCII table, each lowercase letter is exactly 32 positions after its uppercase equivalent. So any character found in the range 'a' to 'z' is converted to uppercase simply by subtracting 32 from its ASCII value, directly modifying the character array in place.

Problem 17: Remove All Spaces from a String

Problem Statement:

Write a C program to remove all spaces from a given string.

Solution:

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

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

    for (i = 0; str[i] != '\0'; i++)
    {
        if (str[i] != ' ' && str[i] != '\n')
        {
            result[j] = str[i];
            j++;
        }
    }
```

```
        result[j] = '\0';

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

Sample Output:

```
Enter a string: Web By Arun
String without spaces: WebByArun
```

Explanation:

The loop walks through every character of str, and only copies a character into result when it is not a space (and not the trailing newline left by fgets()). j keeps track of the next free position in result, and a null terminator is added once copying is complete.

Problem 18: Check Whether a String Contains Only Digits

Problem Statement:

Write a C program to check whether a given string consists only of digit characters (0-9).

Solution:

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

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

    for (i = 0; str[i] != '\0'; i++)
    {
        if (str[i] < '0' || str[i] > '9')
        {
            isNumeric = 0;
            break;
        }
    }

    if (isNumeric)
        printf("The string contains only digits.\n");
    else
        printf("The string contains non-digit characters.\n");

    return 0;
}
```

Sample Output:

```
Enter a string: 20458
The string contains only digits.
```

Explanation:

Each character is compared against the range '0' to '9'. As soon as a character falls outside this range, isNumeric is set to 0 and break exits the loop immediately, since a single non-digit character is enough to disqualify the whole string.