Programming in C

Practice Problems with Solutions — Part 2

Chapter: Decision Control and Looping Statements — For Engineering / Polytechnic Students

This is a continuation of the practice problem set, numbered 19 onward so it can be used alongside Part 1. It covers additional decision-making, looping, array, and string problems, including a menu-driven program, sorting, and further string manipulation exercises.

Section A: Decision Making and Branching Statements

Problem 19: Check Leap Year

Problem Statement:

Write a C program to check whether a given year is a leap year, using logical operators inside an if-else statement.

Solution:

```
#include <stdio.h>
int main()
{
    int year;
    printf("Enter a year: ");
    scanf("%d", &year);

    if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0))
        printf("%d is a leap year.\n", year);
    else
        printf("%d is not a leap year.\n", year);

    return 0;
}
```

Sample Output:

```
Enter a year: 2024
2024 is a leap year.
```

Explanation:

A year is a leap year if it is divisible by 4 but not by 100, or if it is divisible by 400. The logical AND (&&) and OR (||) operators combine these three conditions into a single if statement.

Problem 20: Menu-Driven Calculator (switch inside a loop)

Problem Statement:

Write a C program that repeatedly displays a menu of arithmetic operations, performs the operation chosen using a switch statement, and keeps repeating until the user chooses to exit — combining a loop with a switch statement.

Solution:

```
#include <stdio.h>
```

```c
int main()
{
    int choice;
    double a, b;

    do
    {
        printf("\n1. Add  2. Subtract  3. Multiply  4. Divide  5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        if (choice >= 1 && choice <= 4)
        {
            printf("Enter two numbers: ");
            scanf("%lf %lf", &a, &b);
        }

        switch (choice)
        {
            case 1:
                printf("Result = %.2lf\n", a + b);
                break;
            case 2:
                printf("Result = %.2lf\n", a - b);
                break;
            case 3:
                printf("Result = %.2lf\n", a * b);
                break;
            case 4:
                if (b != 0)
                    printf("Result = %.2lf\n", a / b);
                else
                    printf("Error: Division by zero.\n");
                break;
            case 5:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice.\n");
        }
    } while (choice != 5);

    return 0;
}
```

Sample Output:

```
1. Add  2. Subtract  3. Multiply  4. Divide  5. Exit
Enter your choice: 1
Enter two numbers: 10 5
Result = 15.00
```

Explanation:

The do-while loop, being exit controlled, always shows the menu at least once and keeps repeating until choice equals 5. Inside each pass, the switch statement selects and performs the requested operation, so the loop and the switch work together to create an interactive menu.

Section B: Loop Statements

Problem 21: Factorial of a Number

Problem Statement:

Write a C program to calculate the factorial of a given number using a for loop.

Solution:

```
#include <stdio.h>
int main()
{
    int n, i;
    long long fact = 1;

    printf("Enter a number: ");
    scanf("%d", &n);

    for (i = 1; i <= n; i++)
        fact = fact * i;

    printf("Factorial of %d = %lld\n", n, fact);
    return 0;
}
```

Sample Output:

```
Enter a number: 6
Factorial of 6 = 720
```

Explanation:

fact is initialized to 1 and multiplied by every integer from 1 to n in turn. long long is used for fact because factorial values grow very quickly and can easily exceed the range of a normal int.

Problem 22: Fibonacci Series up to N Terms

Problem Statement:

Write a C program to print the Fibonacci series up to N terms using a while loop.

Solution:

```
#include <stdio.h>
int main()
{
    int n, count = 0, next;
    int t1 = 0, t2 = 1;

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

    printf("Fibonacci Series: ");
    while (count < n)
    {
        printf("%d ", t1);
        next = t1 + t2;
        t1 = t2;
        t2 = next;
        count++;
    }
```

```
    printf("\n");
    return 0;
}
```

Sample Output:

```
Enter number of terms: 7
Fibonacci Series: 0 1 1 2 3 5 8
```

Explanation:

t1 and t2 hold the two most recent terms of the series. On each pass, t1 is printed, the next term is computed as their sum, and both variables are shifted forward, while count keeps track of how many terms have been printed so the while loop knows when to stop.

Problem 23: Multiplication Table of a Number

Problem Statement:

Write a C program to print the multiplication table of a given number from 1 to 10 using a for loop.

Solution:

```
#include <stdio.h>
int main()
{
    int num, i;
    printf("Enter a number: ");
    scanf("%d", &num);

    for (i = 1; i <= 10; i++)
        printf("%d x %d = %d\n", num, i, num * i);

    return 0;
}
```

Sample Output:

```
Enter a number: 7
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
...
7 x 10 = 70
```

Explanation:

The for loop runs the counter i from 1 to 10, and on each pass prints one line of the table by multiplying num with the current value of i.

Problem 24: Reverse the Digits of a Number

Problem Statement:

Write a C program to reverse the digits of a given integer using a while loop.

Solution:

```
#include <stdio.h>
int main()
{
    int num, digit, reversed = 0;
    printf("Enter a number: ");
    scanf("%d", &num);
```

```
    while (num != 0)
    {
        digit = num % 10;
        reversed = reversed * 10 + digit;
        num = num / 10;
    }

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

Sample Output:

```
Enter a number: 4562
Reversed number = 2654
```

Explanation:

On each pass, the last digit of num is extracted with % 10, and reversed is rebuilt by shifting its existing digits one place left (multiplying by 10) before adding the new digit, effectively reconstructing the number in reverse order.

Section C: Arrays

Problem 25: 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 26: 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 27: 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.

Section D: Strings

Problem 28: 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 29: 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 30: 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.