Programming in C

Practice Problems with Solutions — Part 5

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

This is a further continuation of the practice problem set, numbered 55 onward so it can be used alongside Parts 1 to 4. It covers additional decision-making, looping, array, and string problems, including BMI classification, LCM, array rotation, and further string-processing exercises.

Section A: Decision Making and Branching Statements

Problem 55: BMI Category Calculator

Problem Statement:

Write a C program that reads a person's weight (in kg) and height (in metres), calculates the Body Mass Index (BMI), and prints the category using an else-if ladder: BMI below 18.5 - Underweight, 18.5 to 24.9 - Normal, 25 to 29.9 - Overweight, 30 and above - Obese.

Solution:

```
#include <stdio.h>
int main()
{
    float weight, height, bmi;

    printf("Enter weight (kg) and height (m): ");
    scanf("%f %f", &weight, &height);

    bmi = weight / (height * height);
    printf("BMI = %.2f\n", bmi);

    if (bmi < 18.5)
        printf("Category: Underweight\n");
    else if (bmi < 25)
        printf("Category: Normal\n");
    else if (bmi < 30)
        printf("Category: Overweight\n");
    else
        printf("Category: Obese\n");

    return 0;
}
```

Sample Output:

```
Enter weight (kg) and height (m): 68 1.72
BMI = 22.99
Category: Normal
```

Explanation:

The BMI is calculated using the standard formula weight divided by height squared. The else-if ladder then checks the calculated value against each range boundary in increasing order, printing the category for the first range it satisfies.

Problem 56: Simple ATM PIN Validation

Problem Statement:

Write a C program that simulates checking a 4-digit ATM PIN against a stored correct PIN, allowing up to 3 attempts using nested if-else and a loop.

Solution:

```
#include <stdio.h>
int main()
{
    int correctPin = 4521, enteredPin, attempt;
    int accessGranted = 0;

    for (attempt = 1; attempt <= 3; attempt++)
    {
        printf("Attempt %d - Enter PIN: ", attempt);
        scanf("%d", &enteredPin);

        if (enteredPin == correctPin)
        {
            accessGranted = 1;
            break;
        }
        else
            printf("Incorrect PIN.\n");
    }

    if (accessGranted)
        printf("Access granted.\n");
    else
        printf("Access blocked. Too many incorrect attempts.\n");

    return 0;
}
```

Sample Output:

```
Attempt 1 - Enter PIN: 1234
Incorrect PIN.
Attempt 2 - Enter PIN: 4521
Access granted.
```

Explanation:

The for loop allows a maximum of 3 attempts. The if-else inside it checks the entered PIN against the stored PIN on every attempt; a correct match sets accessGranted and immediately breaks out of the loop, so no further attempts are wasted once access has been granted.

Section B: Loop Statements

Problem 57: Digital Root of a Number

Problem Statement:

Write a C program to find the digital root of a number — the single digit obtained by repeatedly summing its digits until only one digit remains — using nested while loops.

Solution:

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

    while (num >= 10)
    {
        sum = 0;
        while (num != 0)
        {
            sum = sum + (num % 10);
            num = num / 10;
        }
        num = sum;
    }

    printf("Digital root = %d\n", num);
    return 0;
}
```

Sample Output:

```
Enter a number: 9875
Digital root = 2
```

Explanation:

The inner while loop adds up the digits of num, exactly like a single-pass digit-sum routine. The outer while loop then repeats this process on the resulting sum as long as it still has more than one digit, until num is finally reduced to a single digit.

Problem 58: Print All Prime Numbers Between 1 and N

Problem Statement:

Write a C program to print all prime numbers between 1 and a given number N, using nested loops.

Solution:

```
#include <stdio.h>
int main()
{
    int n, i, j, isPrime;
    printf("Enter N: ");
    scanf("%d", &n);

    printf("Prime numbers up to %d: ", n);
    for (i = 2; i <= n; i++)
    {
        isPrime = 1;
        for (j = 2; j <= i / 2; j++)
        {
            if (i % j == 0)
```

```
            {
                isPrime = 0;
                break;
            }
        }
        if (isPrime)
            printf("%d ", i);
    }
    printf("\n");
    return 0;
}
```

Sample Output:

```
Enter N: 20
Prime numbers up to 20: 2 3 5 7 11 13 17 19
```

Explanation:

The outer loop runs through every candidate number i from 2 to N. For each candidate, the inner loop checks for a divisor between 2 and i/2; break exits early as soon as one is found, and i is printed only when isPrime remains 1 after the inner loop finishes.

Problem 59: Print a Diamond Pattern

Problem Statement:

Write a C program to print a diamond-shaped pattern of stars using nested loops, for a given number of rows in the upper half.

Solution:

```
#include <stdio.h>
int main()
{
    int n, i, j, space;
    printf("Enter number of rows for half the diamond: ");
    scanf("%d", &n);

    // upper half, including the middle row
    for (i = 1; i <= n; i++)
    {
        for (space = 1; space <= n - i; space++)
            printf("  ");
        for (j = 1; j <= (2 * i - 1); j++)
            printf("* ");
        printf("\n");
    }

    // lower half
    for (i = n - 1; i >= 1; i--)
    {
        for (space = 1; space <= n - i; space++)
            printf("  ");
        for (j = 1; j <= (2 * i - 1); j++)
            printf("* ");
        printf("\n");
    }
    return 0;
}
```

Sample Output:

```
Enter number of rows for half the diamond: 3
    *
  * * *
* * * * *
  * * *
    *
```

Explanation:

The diamond is built as an upper half that widens row by row and a lower half that narrows again. In each row, one inner loop prints the leading spaces needed to centre the stars, and another prints an odd number of stars (2i - 1), which is what creates the diamond's pointed top and bottom.

Problem 60: LCM of Two Numbers

Problem Statement:

Write a C program to find the Least Common Multiple (LCM) of two numbers using a while loop.

Solution:

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

    max = (a > b) ? a : b;

    while (1)
    {
        if (max % a == 0 && max % b == 0)
        {
            printf("LCM = %d\n", max);
            break;
        }
        max++;
    }
    return 0;
}
```

Sample Output:

```
Enter two numbers: 4 6
LCM = 12
```

Explanation:

max starts at the larger of the two numbers, since the LCM can never be smaller than that. The while(1) loop then keeps incrementing max and testing it against both numbers until it finds the first value that both a and b divide evenly, at which point break stops the loop.

Section C: Arrays

Problem 61: 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 62: 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 63: 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.

Section D: Strings

Problem 64: 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 65: 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 66: 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, k, 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.