Programming in C

Practice Problems with Solutions — Part 3

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

This is a further continuation of the practice problem set, numbered 31 onward so it can be used alongside Parts 1 and 2. It covers additional decision-making, looping, array, and string problems, including array insertion and deletion, diagonal sums, and further string-processing exercises.

Section A: Decision Making and Branching Statements

Problem 31: Vowel or Consonant Check Using switch

Problem Statement:

Write a C program to check whether a given alphabet is a vowel or a consonant using a switch statement.

Solution:

```
#include <stdio.h>
int main()
{
    char ch;
    printf("Enter an alphabet: ");
    scanf("%c", &ch);

    switch (ch)
    {
        case 'a': case 'e': case 'i': case 'o': case 'u':
        case 'A': case 'E': case 'I': case 'O': case 'U':
            printf("%c is a vowel.\n", ch);
            break;
        default:
            printf("%c is a consonant.\n", ch);
    }
    return 0;
}
```

Sample Output:

```
Enter an alphabet: o
o is a vowel.
```

Explanation:

Multiple case labels are stacked one after another with no statements or break between them, so all ten vowel cases (uppercase and lowercase) fall through into the same printf() statement. Any character that does not match one of these labels is caught by the default case.

Problem 32: Check Whether a Triangle is Valid

Problem Statement:

Write a C program that reads the three sides of a triangle and checks, using logical conditions inside an if-else statement, whether they can form a valid triangle.

Solution:

```
#include <stdio.h>
int main()
{
    float a, b, c;
    printf("Enter three sides of the triangle: ");
    scanf("%f %f %f", &a, &b, &c);

    if (a > 0 && b > 0 && c > 0 &&
        (a + b > c) && (b + c > a) && (a + c > b))
        printf("The triangle is valid.\n");
    else
        printf("The triangle is not valid.\n");

    return 0;
}
```

Sample Output:

```
Enter three sides of the triangle: 3 4 5
The triangle is valid.
```

Explanation:

A triangle is valid only when all three sides are positive and the sum of any two sides is strictly greater than the third side (the triangle inequality). All three inequality checks are joined using the logical AND (&&) operator inside a single if condition.

Section B: Loop Statements

Problem 33: GCD of Two Numbers

Problem Statement:

Write a C program to find the Greatest Common Divisor (GCD) of two numbers using a while loop and the Euclidean algorithm.

Solution:

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

    while (b != 0)
    {
        temp = b;
        b = a % b;
        a = temp;
    }

    printf("GCD = %d\n", a);
    return 0;
}
```

Sample Output:

```
Enter two numbers: 48 18
```

```
GCD = 6
```

Explanation:

On each pass, b is replaced by the remainder of a divided by b, and a takes the previous value of b. This is repeated until b becomes zero, at which point a holds the GCD, following the Euclidean algorithm.

Problem 34: Check Whether a Number is an Armstrong Number

Problem Statement:

Write a C program to check whether a given 3-digit number is an Armstrong number, that is, the sum of the cubes of its digits equals the number itself.

Solution:

```
#include <stdio.h>
int main()
{
    int num, original, digit, sum = 0;
    printf("Enter a 3-digit number: ");
    scanf("%d", &num);
    original = num;

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

    if (sum == original)
        printf("%d is an Armstrong number.\n", original);
    else
        printf("%d is not an Armstrong number.\n", original);

    return 0;
}
```

Sample Output:

```
Enter a 3-digit number: 153
153 is an Armstrong number.
```

Explanation:

The original value is preserved in original before num is broken down digit by digit inside the while loop. Each digit is cubed and added to sum, and the final sum is compared against the original number to decide the result.

Problem 35: Number Pyramid Pattern (Nested Loops)

Problem Statement:

Write a C program to print a number pyramid pattern, where each row prints numbers from 1 up to the row number, using nested loops.

Solution:

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

```
    int rows, i, j;
    printf("Enter number of rows: ");
    scanf("%d", &rows);

    for (i = 1; i <= rows; i++)
    {
        for (j = 1; j <= i; j++)
            printf("%d ", j);
        printf("\n");
    }
    return 0;
}
```

Sample Output:

```
Enter number of rows: 4
1
1 2
1 2 3
1 2 3 4
```

Explanation:

The outer loop selects the row, and the inner loop prints the numbers 1 through i for that row, so each successive row has exactly one more number than the row before it.

Problem 36: Sum of Even and Odd Numbers Separately

Problem Statement:

Write a C program to find the sum of all even numbers and the sum of all odd numbers from 1 to N, using a for loop with an if-else check.

Solution:

```
#include <stdio.h>
int main()
{
    int n, i, evenSum = 0, oddSum = 0;
    printf("Enter N: ");
    scanf("%d", &n);

    for (i = 1; i <= n; i++)
    {
        if (i % 2 == 0)
            evenSum = evenSum + i;
        else
            oddSum = oddSum + i;
    }

    printf("Sum of even numbers = %d\n", evenSum);
    printf("Sum of odd numbers = %d\n", oddSum);
    return 0;
}
```

Sample Output:

```
Enter N: 10
Sum of even numbers = 30
Sum of odd numbers = 25
```

Explanation:

Inside the for loop, the modulus operator (% 2) tests whether the current value of i is even or odd, and the if-else statement routes each number into the correct running total.

Section C: Arrays

Problem 37: 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 38: 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.

Problem 39: Sum of Diagonal Elements of a Square Matrix

Problem Statement:

Write a C program to find the sum of the principal diagonal elements of a square (N x N) matrix.

Solution:

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

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

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

    for (i = 0; i < n; i++)
        sum = sum + a[i][i];
```

```
    printf("Sum of diagonal elements = %d\n", sum);
    return 0;
}
```

Sample Output:

```
Enter size of the square matrix: 3
Sum of diagonal elements = 15
```

Explanation:

The principal diagonal of a square matrix consists of elements whose row index equals their column index, that is, a[0][0], a[1][1], a[2][2], and so on. A single loop with a[i][i] is enough to add up exactly these elements.

Section D: Strings

Problem 40: Count the Number of Words in a Sentence

Problem Statement:

Write a C program to count the number of words in a sentence, assuming words are separated by single spaces.

Solution:

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

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

    for (i = 0; str[i] != '\0'; i++)
    {
        if (str[i] == ' ' && str[i + 1] != ' ' && str[i + 1] != '\0')
            wordCount++;
    }
    wordCount++;   // last word has no trailing space before it

    printf("Number of words = %d\n", wordCount);
    return 0;
}
```

Sample Output:

```
Enter a sentence: Programming in C is fun
Number of words = 5
```

Explanation:

fgets() is used instead of scanf("%s", ...) so that the full sentence, including spaces, is read as one line. The loop counts a new word every time a space is followed by a non-space character, and one is added at the end to account for the final word.

Problem 41: 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 42: 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.