Programming in C

Practice Problems with Solutions

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

This booklet contains solved programming problems covering decision-making statements, loops, arrays, and strings. 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: Decision Making and Branching Statements

Problem 1: Check Positive, Negative or Zero

Problem Statement:

Write a C program to read an integer and check whether it is positive, negative, or zero.

Solution:

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

    if (num > 0)
        printf("The number is positive.\n");
    else if (num < 0)
        printf("The number is negative.\n");
    else
        printf("The number is zero.\n");

    return 0;
}
```

Sample Output:

```
Enter a number: -7
The number is negative.
```

Explanation:

The program uses an else-if ladder to test three mutually exclusive conditions. Only one branch executes, based on the value stored in num.

Problem 2: Largest of Three Numbers

Problem Statement:

Write a C program to find the largest among three numbers entered by the user, using nested if-else.

Solution:

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

```
{
    int a, b, c;
    printf("Enter three numbers: ");
    scanf("%d %d %d", &a, &b, &c);

    if (a >= b)
    {
        if (a >= c)
            printf("Largest number is %d\n", a);
        else
            printf("Largest number is %d\n", c);
    }
    else
    {
        if (b >= c)
            printf("Largest number is %d\n", b);
        else
            printf("Largest number is %d\n", c);
    }
    return 0;
}
```

Sample Output:

```
Enter three numbers: 12 45 30
Largest number is 45
```

Explanation:

The outer if-else first narrows the comparison to a vs b, and the inner if-else then compares the larger of those two against c, giving the overall largest value using nested decision making.

Problem 3: Grade Calculator

Problem Statement:

Write a C program that reads the marks obtained by a student (out of 100) and prints the grade using the following scheme: 90 and above -> A, 75-89 -> B, 60-74 -> C, 40-59 -> D, below 40 -> F.

Solution:

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

    if (marks >= 90)
        printf("Grade: A\n");
    else if (marks >= 75)
        printf("Grade: B\n");
    else if (marks >= 60)
        printf("Grade: C\n");
    else if (marks >= 40)
        printf("Grade: D\n");
    else
        printf("Grade: F\n");

    return 0;
```

```
}
```

Sample Output:

```
Enter marks: 82
Grade: B
```

Explanation:

This is a classic use of the else-if ladder, where conditions are checked in decreasing order of marks. As soon as one range matches, its grade is printed and the remaining checks are skipped.

Problem 4: Simple Calculator Using switch

Problem Statement:

Write a C program that reads two numbers and an operator (+, -, *, /), and uses a switch statement to perform the corresponding arithmetic operation.

Solution:

```
#include <stdio.h>
int main()
{
    double a, b, result;
    char op;

    printf("Enter first number, operator, second number: ");
    scanf("%lf %c %lf", &a, &op, &b);

    switch (op)
    {
        case '+':
            result = a + b;
            printf("Result = %.2lf\n", result);
            break;
        case '-':
            result = a - b;
            printf("Result = %.2lf\n", result);
            break;
        case '*':
            result = a * b;
            printf("Result = %.2lf\n", result);
            break;
        case '/':
            if (b != 0)
            {
                result = a / b;
                printf("Result = %.2lf\n", result);
            }
            else
                printf("Error: Division by zero.\n");
            break;
        default:
            printf("Invalid operator.\n");
    }
    return 0;
}
```

Sample Output:

```
Enter first number, operator, second number: 15 * 4
```

```
Result = 60.00
```

Explanation:

The switch statement compares the character stored in op against each case label. A break after every case prevents fall-through, and the default case safely handles any operator that is not one of the four expected symbols.

Section B: Loop Statements

Problem 5: Sum and Average of First N Natural Numbers (for loop)

Problem Statement:

Write a C program to calculate the sum and average of the first N natural numbers using a for loop.

Solution:

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

    for (i = 1; i <= n; i++)
        sum = sum + i;

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

    return 0;
}
```

Sample Output:

```
Enter N: 5
Sum = 15
Average = 3.00
```

Explanation:

The for loop initializes i to 1, checks i <= n before every pass, and increments i after each pass, accumulating the running total in sum until all N numbers have been added.

Problem 6: Sum of Digits of a Number (while loop)

Problem Statement:

Write a C program to find the sum of digits of a given number using a while loop.

Solution:

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

    while (num != 0)
```

```
    {
        digit = num % 10;
        sum = sum + digit;
        num = num / 10;
    }

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

Sample Output:

```
Enter a number: 4582
Sum of digits = 19
```

Explanation:

On each pass, the last digit is extracted using the modulus operator (% 10) and added to sum, and the number is then reduced by integer division (/ 10). The while loop, an entry controlled loop, continues until num becomes zero.

Problem 7: Check Whether a Number is Prime (for loop with break)

Problem Statement:

Write a C program to check whether a given number is prime, using a for loop with a break statement.

Solution:

```
#include <stdio.h>
int main()
{
    int n, i, isPrime = 1;
    printf("Enter a number: ");
    scanf("%d", &n);

    if (n <= 1)
        isPrime = 0;

    for (i = 2; i <= n / 2; i++)
    {
        if (n % i == 0)
        {
            isPrime = 0;
            break;    // no need to check further once a factor is found
        }
    }

    if (isPrime)
        printf("%d is a prime number.\n", n);
    else
        printf("%d is not a prime number.\n", n);

    return 0;
}
```

Sample Output:

```
Enter a number: 29
29 is a prime number.
```

Explanation:

The loop tests possible divisors from 2 up to n/2. As soon as a divisor is found, isPrime is set to 0 and break immediately exits the loop, since continuing to check further divisors would be unnecessary.

Problem 8: Sum of Natural Numbers Skipping Multiples of 3 (continue statement)

Problem Statement:

Write a C program to find the sum of all natural numbers from 1 to N, skipping any number that is a multiple of 3, using the continue statement.

Solution:

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

    for (i = 1; i <= n; i++)
    {
        if (i % 3 == 0)
            continue;   // skip multiples of 3
        sum = sum + i;
    }

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

Sample Output:

```
Enter N: 10
Sum = 37
```

Explanation:

Whenever i is a multiple of 3, continue skips the sum = sum + i statement for that pass and jumps straight to the update expression (i++), so multiples of 3 never get added to the total.

Problem 9: Input Validation Using a Conditional goto

Problem Statement:

Write a C program that repeatedly asks the user to enter a positive number, using a conditional goto statement to re-ask if the entered value is not positive.

Solution:

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

    reenter:
        printf("Enter a positive number: ");
        scanf("%d", &num);

    if (num <= 0)
        goto reenter;   // conditional goto: jump only if input is invalid
```

```
    printf("You entered a valid positive number: %d\n", num);
    return 0;
}
```

Sample Output:

```
Enter a positive number: -3
Enter a positive number: 0
Enter a positive number: 8
You entered a valid positive number: 8
```

Explanation:

The label reenter marks the point to jump back to. Since the goto statement is inside an if block, the jump is conditional and occurs only when num is not a positive number; the same task could also be written more cleanly using a do-while loop.

Problem 10: Right-Angled Triangle Pattern (Nested Loops)

Problem Statement:

Write a C program to print a right-angled triangle pattern of stars using nested loops, for a given number of rows.

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

Sample Output:

```
Enter number of rows: 4
*
* *
* * *
* * * *
```

Explanation:

The outer loop controls the row number, while the inner loop runs from 1 to the current row number i, printing one extra star per row. This is a typical application of nested loops for pattern generation.

Section C: Arrays

Problem 11: 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 12: 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 13: 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 14: 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.

Section D: Strings

Problem 15: 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 16: 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 17: 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 18: 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.