ASSIGNMENT 03
PROGRAMMING FUNDAMENTALS

INSTRUCTIONS:

1) Plagiarism is strictly forbidden. Each individual must complete the work independently as it is part of the learning process.
2) The naming format must be strictly followed as given in the instruction note:

23P-0603_Aimalkhan_Q1.c
23P-0603_Aimalkhan_Q2.c
23P-0603_Aimalkhan_Q3.c
23P-0603_Aimalkhan_Q4.c

3) Anyone who fails to follow the naming format will receive zero marks.
4) Everything covered in class so far is included in the assignment, so don't go beyond that.
5) Comment your code properly.
6) Upload four .c files.
7) Anyone failed to follow these instructions will get a straight zero.
8) Don't use AI for this assignment try to do it on your own it will be helpful in your sessional exam.

Problem 1:

You are tasked with writing a program that finds and prints a special sequence of integers in reverse order using while loop, following these rules:

Starting number: The user inputs a positive integer N.

Reduction process: Starting from N, repeatedly do the following: If the number is even, divide it by 2. If the number is odd, multiply it by 3 and add 1. Continue until the number reaches 1.

Reversing the sequence: Print all the numbers encountered during this process in reverse order (i.e., starting from 1 and ending at N).

Loop Condition: You must use a while loop to implement the process. Do not use any other types of loops.

Challenge: You are not allowed to use arrays, vectors, or any data structures that would directly store the sequence beforehand. Instead, you must find a way to reverse the output through logic.

Sample Output:

```
Enter N: 6
Sequence in reverse order: 1 2 4 8 16 5 10 3 6
```

Understanding sample output: If you start with N = 6, the sequence is generated as follows:

Start with:

6. 6 is even → divide by 2 → result is 3.

3 is odd → multiply by 3 and add 1 → result is 10.

10 is even → divide by 2 → result is 5.

5 is odd → multiply by 3 and add 1 → result is 16.

16 is even → divide by 2 → result is 8.

8 is even → divide by 2 → result is 4.

4 is even → divide by 2 → result is 2.

2 is even → divide by 2 → result is 1.

Problem 2: Advanced Banking System

Write a loop-based banking simulation.

Requirements:

1. Starting Balance: Begin with $5000.

2. Menu Options (in a do-while loop):

1. Deposit money
2. Withdraw money
3. Check balance
4. Transfer money
5. View mini history (only last 3 operations, shown using logic not arrays)
6. Exit

3. Rules:

• Deposit must be positive and not more than 10,000 dollars.

• Withdraw only if balance is enough otherwise, print 'Insufficient funds, 50 dollars fine applied' and deduct 50 dollars.

• Transfer to a hardcoded account balance = 1000 dollars. Deduct 2% transaction fee.

• Keep the last three operations' names (like Deposit, Withdraw, Transfer) using only variables and shifting logic (no arrays).
• After each operation, ask 'Do you want to perform another operation? (y/n).

4. Exit Confirmation:
Ask 'Are you sure you want to exit? (y/n)' before ending the program.

Problem 3: Multi-Layer Hollow Diamond

Write a program that prints a two-layer diamond.

Requirements:

1. Ask the user for:
• Height (odd number ≥ 3)
• Symbol to print (e.g., * or #)
• Whether to make it hollow or filled (y = hollow, n = filled)
• Whether to print one diamond or two overlapping diamonds (outer and inner).

2. Rules:
1. The height must be height >= 3.
2. Outer and inner diamonds must have different symbols.
3. Make sure to add checks.
4. Use only * / # symbols.
5. For the double diamond, the inner diamond should be smaller and drawn using the same nested loop logic**.**

Diamond Structures (for better understanding):
Filled Diamond (height = 5):

```
  *
 ***
*****
 ***
  *
```

Hollow Diamond (height = 5):

```
  *
 * *
*   *
 * *
  *
```

Double-Layer Diamond (outer is * , inner is #, height = 9): (it can be outer #, inner *)

```
        *
       * # *
      * # # *
     * # # # *
    * # # # # *
     * # # # *
      * # # *
       * # *
        *
```

Problem 4: Advanced Multiplication Table with Skips, Sums, and Primes

Create a loop-based multiplication table program.

Requirements:

1. Ask the user for starting and ending numbers.

Example: start = 3, end = 6 → show tables for 3, 4, 5, 6.

2. Display Rules:

• For each number, print its multiplication table from 1 to 10.

• If any product is divisible by both 3 and 5, print 'SKIP' instead of the number.

• Mark numbers that are prime with a * (you must find primes using nested loops only).

• After each table, print the sum of all products in that table.

• Keep a running total sum of all tables combined and display it at the end.

3. Additional Challenges:

• Track the largest and smallest product using variables only (no arrays).

• If the total sum > 10,000, display 'Overflow Detected'.