**Task 1: Banker's Algorithm Simulation**

**Task Description:**

The goal is to implement a simulation of the Banker's Algorithm in **C** to ensure deadlock avoidance in a system. The program must determine whether the system is in a safe state and handle dynamic resource requests while ensuring the system remains safe.

**Input Requirements:**

* Number of processes and resource types.
* Matrices for Allocation, Max, and Available resources.
* Validation of inputs to prevent errors such as negative values or invalid configurations.

**Output Requirements:**

* Display the system's state (Safe or Unsafe) after every operation.
* Provide detailed feedback about whether granting a resource request keeps the system in a safe state.

**Libraries/Header Files Used:**

* **stdio.h**: Provides functions for input and output operations.
* **stdbool.h**: Enables the use of boolean data types for logical conditions.
* **stdlib.h**: Supports dynamic memory allocation and other utility functions.

**Detailed Explanation:**

* **Input:**
* **Processes and Resources**: The user specifies the number of **processes[]** and types of resources. These values are stored in global variables and arrays for further computation.
* **Resource Matrices**:
  + **allocation[]**: Tracks the resources currently allocated to each process.
  + **max[]**: Represents the maximum resources each process has requested.
  + **need[]**: Calculated as Max - Allocation, representing the remaining resources each process requires to complete.
* **Banker's Algorithm Logic:**
* **Safety Check**:
  + A process can execute if all its resource needs can be satisfied with the currently available resources.
  + After a process executes, its allocated resources are added back to the available pool.
  + This process is repeated until all processes are executed or no further progress can be made.
* **Safe State**:
  + If all processes can execute without leading to a deadlock, the system is in a safe state.
  + If any process cannot execute due to resource unavailability, the system is unsafe.

**Functions Used:**

1. **input()**:
   * Gathers user input for the number of processes, resource types, and matrices.
   * Validates input values and recalculates the Need matrix.
2. **is\_Safe()**:
   * Implements the Banker's Algorithm to determine if the system is in a safe state.
   * Uses a work array to simulate resource allocation and process execution.
   * Flags any processes that cannot complete due to insufficient resources.

**Key Concepts:**

* **need[] Validation**:
  + Ensures that no element in the need**[]** is negative, as this would indicate invalid input.
* **Dynamic Resource Handling**:
  + Simulates changes in resource allocation and ensures that every state transition maintains system safety.

**Program Flow:**

1. The user inputs the number of processes and resource types.
2. Resource matrices (**allocation[]**, **max[]**, and **available[]**) are defined by the user at run time for each **processes[]**
3. The Need matrix is computed as Max - Allocation.
4. The Banker's Algorithm checks for system safety:
   * If the system is safe, a message is displayed.
   * If unsafe, the program halts further operations.
5. The program ends after freeing all dynamically allocated memory.