The given code provides a solution for arranging the sitting scheme of a group of invited guests. The main objective is to seat the guests at different tables while taking into account their dislikes, ensuring that no person sits with someone they dislike.

Approach: The code utilizes a depth-first search (DFS) algorithm to explore different seating arrangements. It starts by selecting a guest from the list of invited guests and recursively tries to seat them at each table. If the selected guest can be seated at a particular table without any conflicts, they are added to that table. The algorithm then proceeds to seat the remaining guests recursively. If at any point a conflict arises and it becomes impossible to seat all guests, the algorithm backtracks and explores alternative arrangements.

Implementation: The code defines a function named dfs that performs the DFS traversal. It takes the list of remaining guests and the tables as parameters. Within the dfs function, a guest is selected from the list of guests and removed from the list. It then iterates through each table and checks if it is safe to seat the guest at that table using the isSafeToSit function. If it is safe, the guest is added to the table, and the dfs function is recursively called to seat the remaining guests. If a successful arrangement is found, the function returns true. If not, the guest is removed from the table (backtracking) to explore other possibilities. Finally, if it is not possible to arrange all the guests at the tables, the function returns false.

The isSafeToSit function checks whether a person can be seated at a table without any conflicts. It iterates through the guests already seated at the table and checks if the person dislikes any of them by using the dislikes mapping.

The seating arrangement is stored in a vector of vectors called tables. Each table represents a group of guests, and the guests are stored in the vector associated with each table.

The seating arrangement is printed using the printSittingArrangement function, which iterates through the tables and prints the guests seated at each table.

Sample Output: The code generates the seating arrangement based on the given guests and their dislikes. The seating arrangement may vary depending on the order of exploration and backtracking during the DFS traversal. One possible seating arrangement is:

Table 1: Frank Eve Charlie Table 2: David Bob Alice

Explanation:

* Frank, Eve, and Charlie are seated together at Table 1. There are no conflicts between these guests.
* David, Bob, and Alice are seated together at Table 2. There are no conflicts between these guests.

Overall, the code provides a solution for arranging the sitting scheme of the guests while considering their dislikes, ensuring a harmonious seating arrangement.