The program is designed to manage a database of hotels using a hash table where each bucket is a binary search tree (BST). This design choice allows for efficient insertion, deletion, and search operations. The hash table provides constant time complexity for these operations on average, while the BSTs handle collisions in the hash table, maintaining the data in a sorted order and providing logarithmic time complexity for these operations in the worst case.

The program consists of several classes:

1.The Entry class is a fundamental part of the program as it encapsulates the data related to a hotel. It is designed to hold all the necessary information about a hotel, which includes the hotel's name, the city where it's located, its star rating, price, the country where it's located, and its address.

The class has private member variables for each of these pieces of information. Making these variables private ensures that they can only be accessed and modified through the class's methods, which helps maintain the integrity of the data.

The key for each entry is a combination of the hotel name and city name. This is a design choice made to ensure that each hotel in a specific city has a unique identifier within the database. This key is used for inserting, searching, and deleting operations in the hash table and BST.

The class provides getter methods (getHotelName(), getCityName(), getStars(), getPrice(), getCountryName(), getAddress()) for each of these variables. These methods are used to access the values of these variables from outside the class.

The toString() method is a utility function that converts the data of an Entry object into a formatted string. This is particularly useful when we need to print or display the hotel information. The method uses a stringstream to format and concatenate the member variables into a single string, which is then returned.

2.The Node class is a fundamental building block of the Binary Search Tree (BST) used in this program. It's a template class used to store hotel data.

Each Node object contains a key, a vector of values, and pointers to its left child, right child, and parent nodes.

The key is a unique identifier for each node and is used to organize the nodes within the BST. In this program, the key is a combination of the hotel name and city name.

The vector of values is used to store the actual data associated with the key. In this case, it's a list of Entry objects, which represent hotels. The reason for using a vector is to handle collisions, i.e., when multiple hotels have the same key (hotel name and city name). Instead of overwriting the existing data or creating a new node with the same key, the program simply adds the new hotel to the vector of values in the existing node.

The left, right, and parent pointers are used to link the node to other nodes in the BST. The left pointer points to the node's left child, the right pointer points to the node's right child, and the parent pointer points to the node's parent. These pointers are essential for navigating the BST and performing operations like insertions, deletions, and searches.

The class provides a constructor that initializes a node with a given key. The left, right, and parent pointers are initially set to nullptr, indicating that the node doesn't have any connections when it's first created.

3. BST: This is a template class that represents a BST. Each BST contains a root node and provides methods for inserting a node, deleting a node, finding a node, performing an in-order traversal, and finding the minimum node. The class also provides a helper function for the in-order traversal.

4. HashTable: This class represents the hash table. It contains an array of BSTs (the buckets), the current size of the hash table, the total capacity of the hash table, and the total number of collisions. The class provides methods for inserting an entry, searching for an entry, finding all entries that match a given city name and star rating, erasing an entry, and dumping the contents of the hash table to a file. It also provides getter methods for the size of the hash table and the number of collisions.

5. main: This is the main function of the program. It creates an instance of the HashTable class and provides a command-line interface for the user to interact with the hash table. The user can insert entries, search for entries, find all entries that match a given city name and star rating, erase entries, and dump the contents of the hash table to a file.

The design of the program allows for efficient management of the hotel database. The use of a hash table with BSTs as buckets provides a balance between speed and memory usage.