UTM

UNIVERSITI TEKNOLOGI MALAYSIA

SECJ2013

DATA STRUCTURE AND ALGORITHM

ASSIGNMENT 2

SECTION 02

LECTURER: Ms. Lizawati binti Mi Yusuf

GROUP NAME: LogiCode

NAME	MATRIC NO.
Ong Yi Yan	A22EC0101
Tang Yan Qing	A22EC0109
Koh Su Xuan	A22EC0060

Table of Contents

Objective

- To simulate an actual Hotel Booking System in a way that administrators can use it to manage the room bookings in a hotel.
- To improve understanding and enhance application of data structure: linked list in a real-world scenario.
- To utilize file mechanism in C++ to read data from external files as a simulation of the database of the system.

Synopsis

The Hotel Booking System is designed for hotel administrators to manage room bookings in a hotel. By using file input operation, the system reads the file containing booking information which are check-in date, check-out date, room number, room type, identification card number and total price of bookings and stores in BookingNode class. With the administrator menu provided, the administrator is able to insert, delete, find, sort and view the booking information stored in the BookingNode class. The menu is further detailed to insert menu and delete menu in which the insert menu involves the insertion of booking data at front, in the middle under certain conditions such as at certain position, before or after certain position, before or after specific booking information like check-in date, check-out date and room number and at the end of the booking data while delete menu involves deleting the first booking, specific position booking, booking with specific check-in date and room number and the last booking data. All menus are operated and implemented with linked list data structure concept and technique.

Design

Class diagram

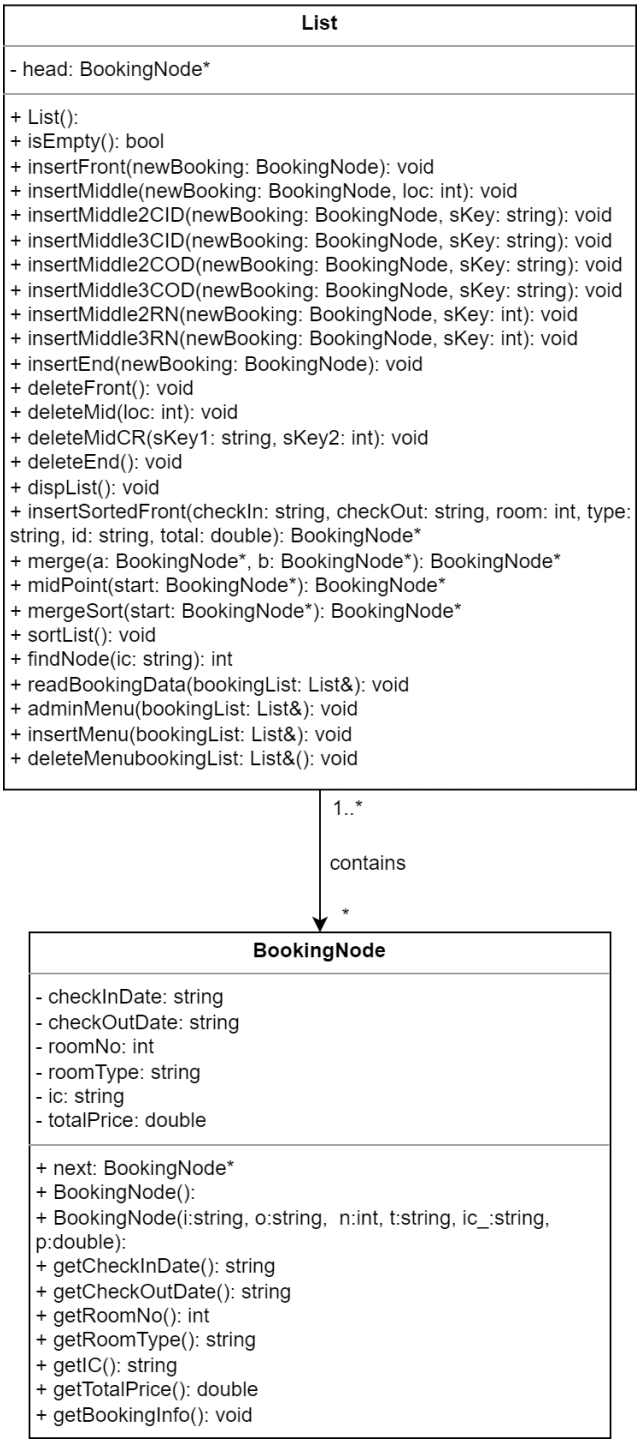

Figure 1: Class Diagram of Hotel Booking System

Flowchart

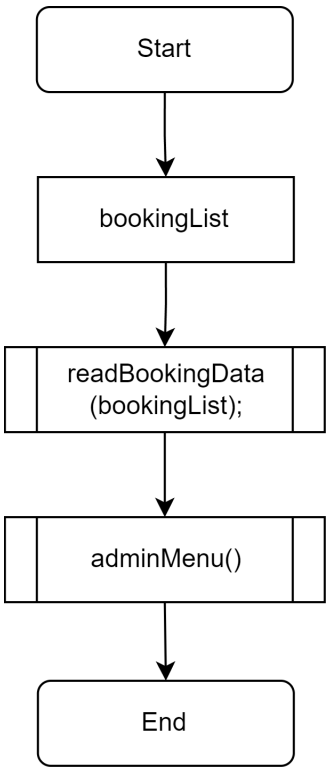

Figure 2: Flowchart of main Function

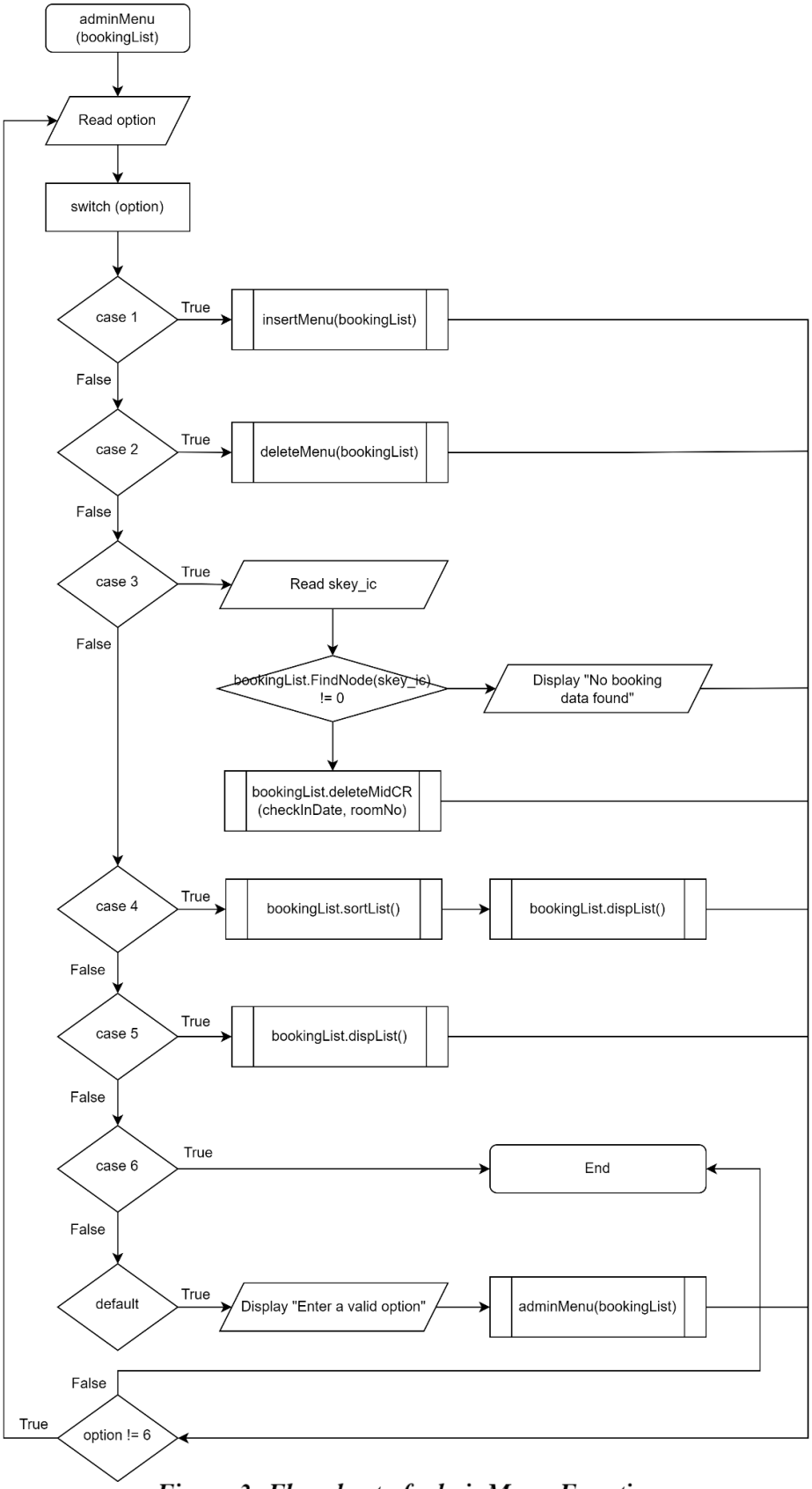

Figure 3: Flowchart of adminMenu Function

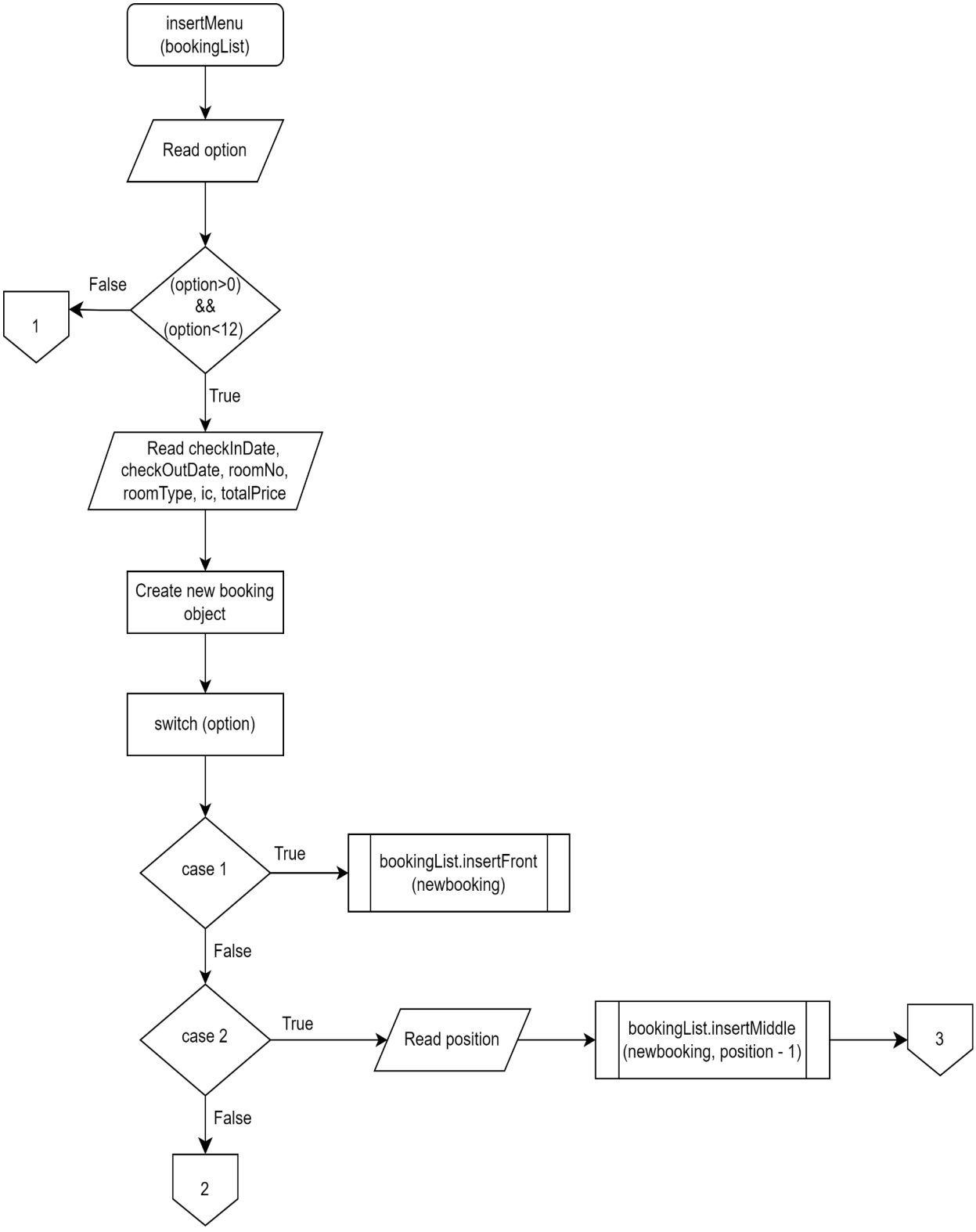

Figure 4.1: Flowchart of insertMenu Function

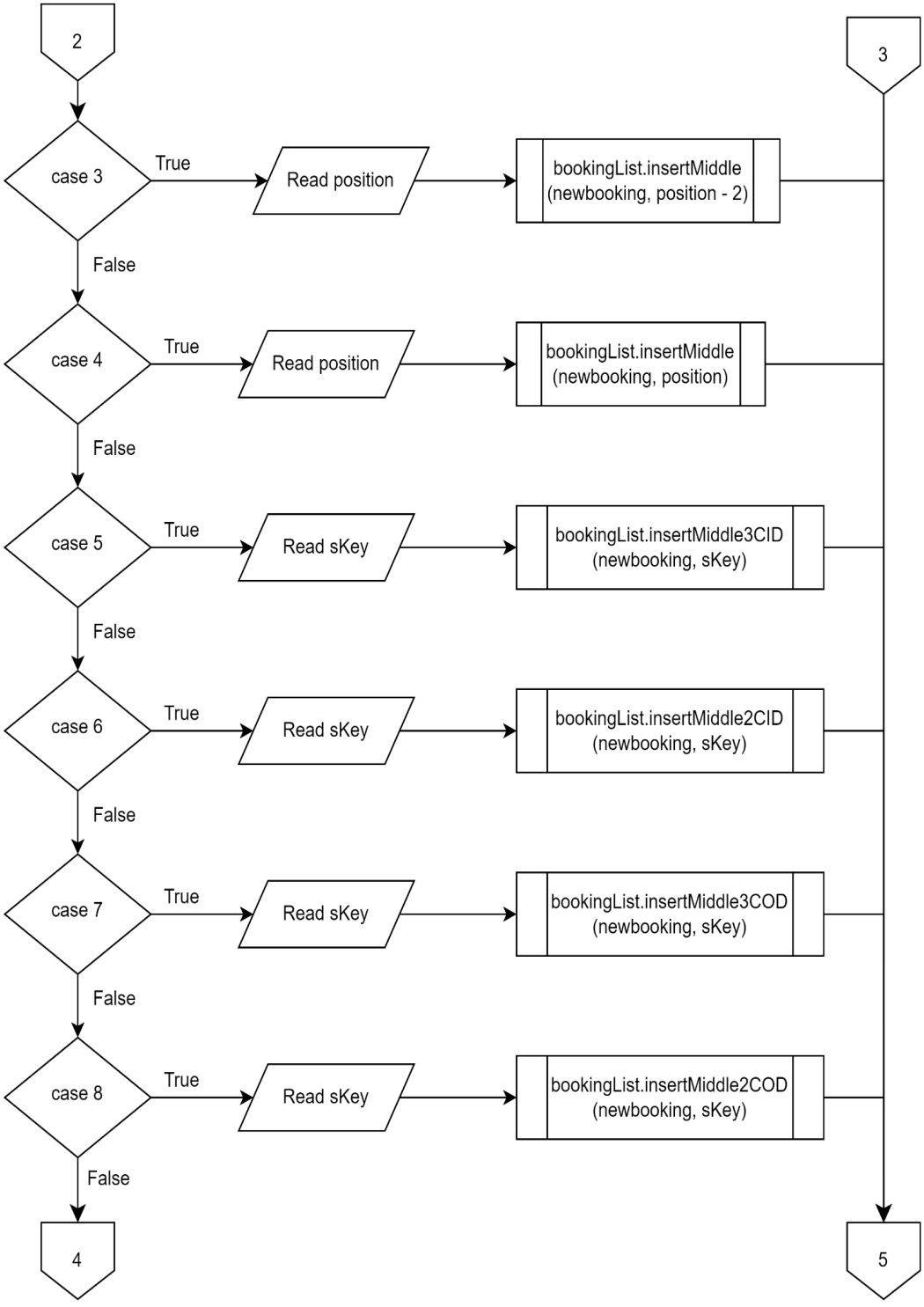

Figure 4.2: Flowchart of insertMenu Function

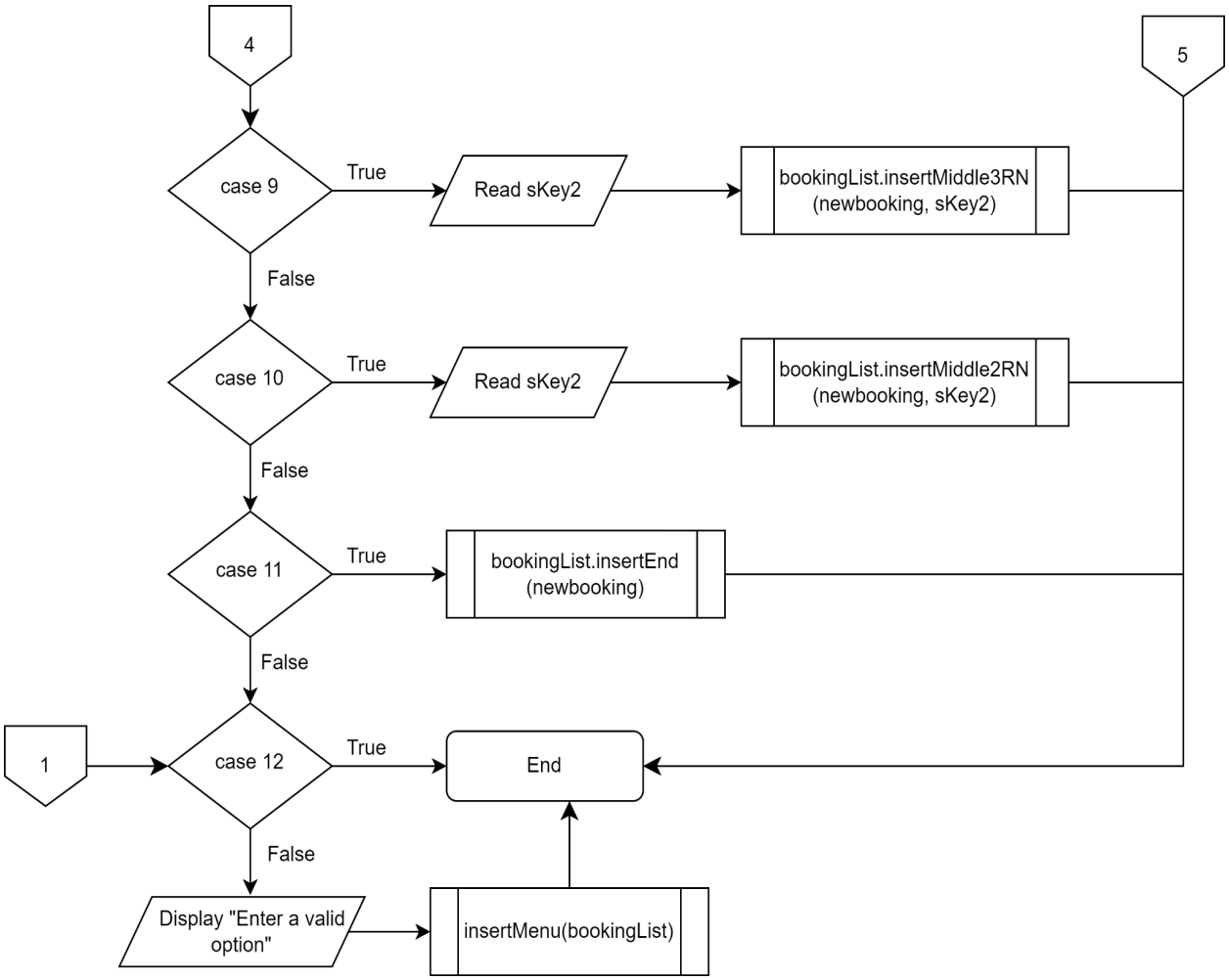

Figure 4.3: Flowchart of insertMenu Function

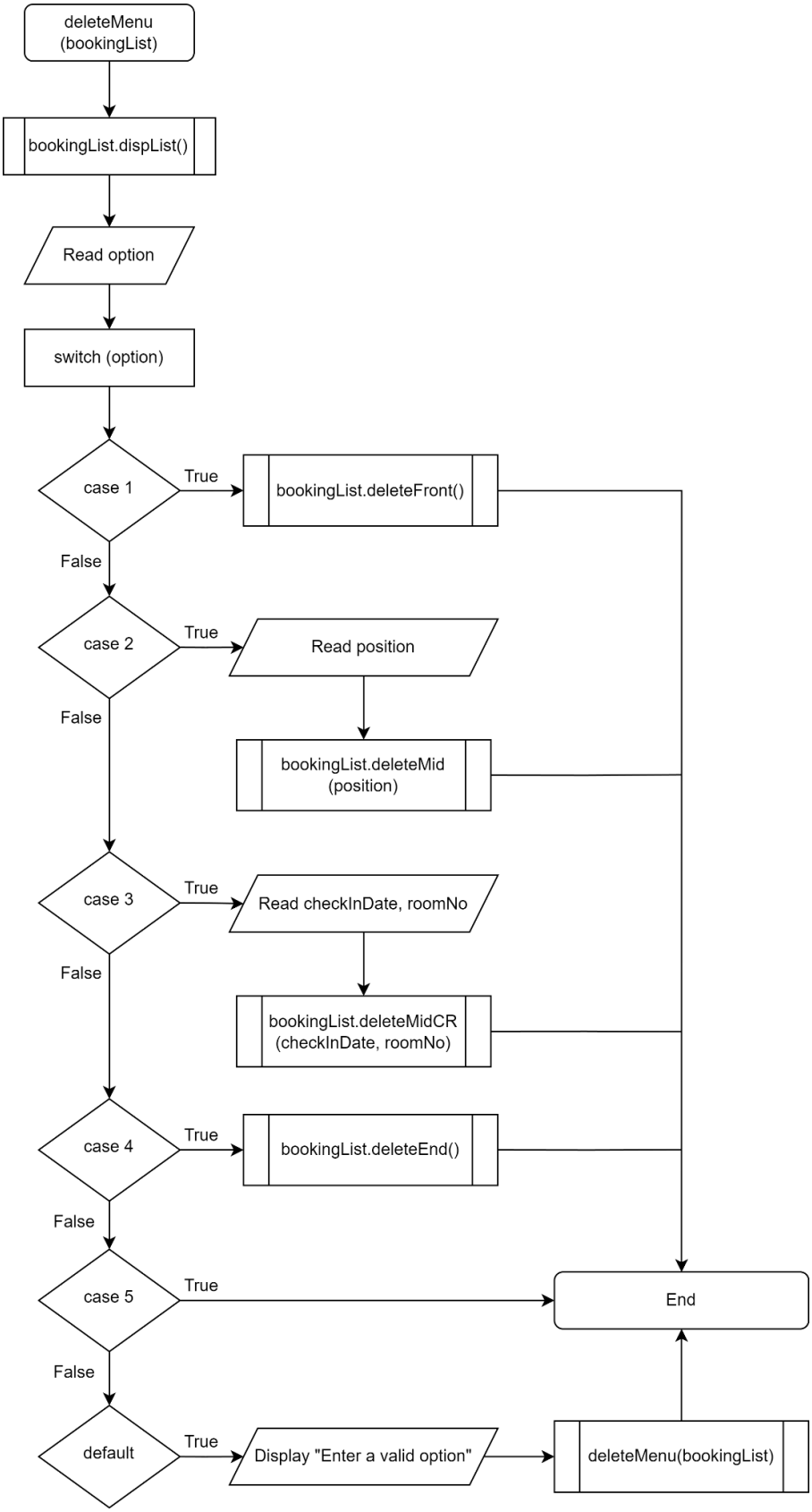

Figure 5: Flowchart of deleteMenu Function

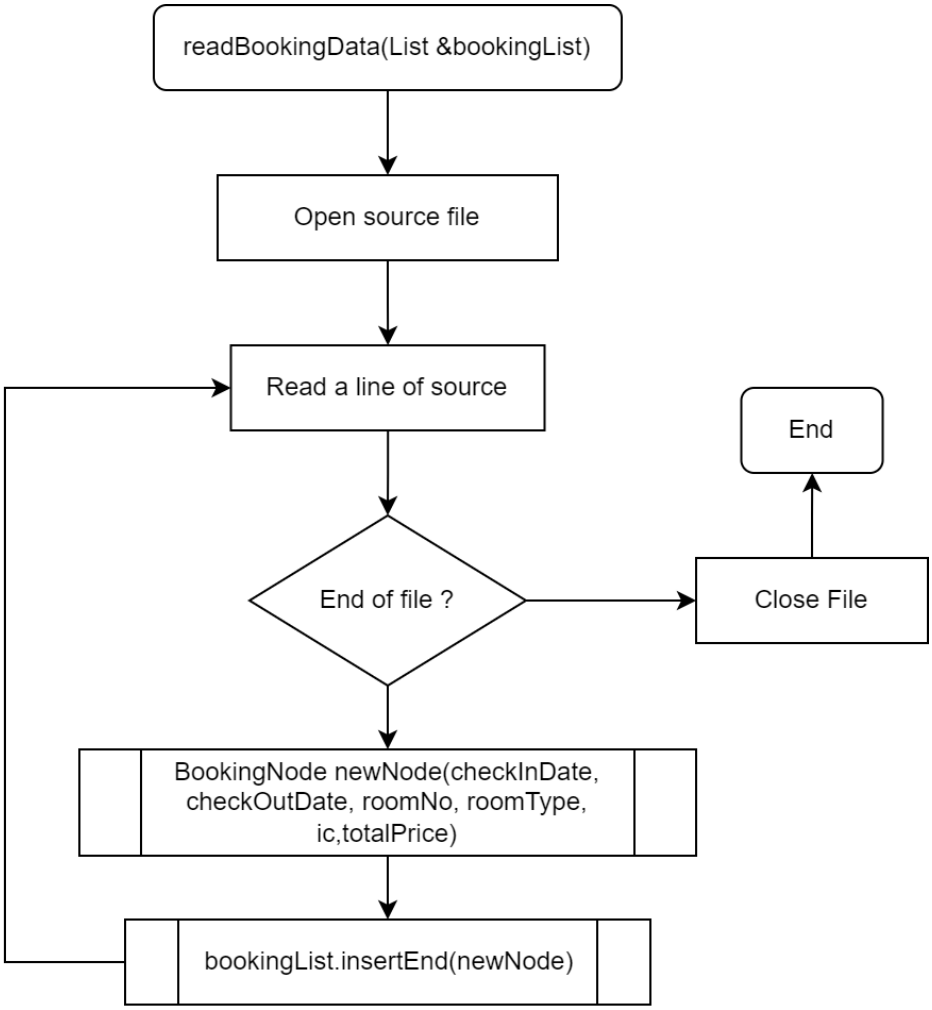

Figure 6: Flowchart of readBookingData Function

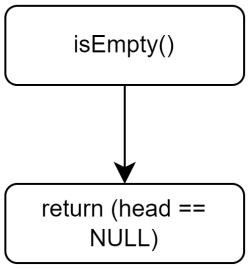

Figure 7: Flowchart of isEmpty Function

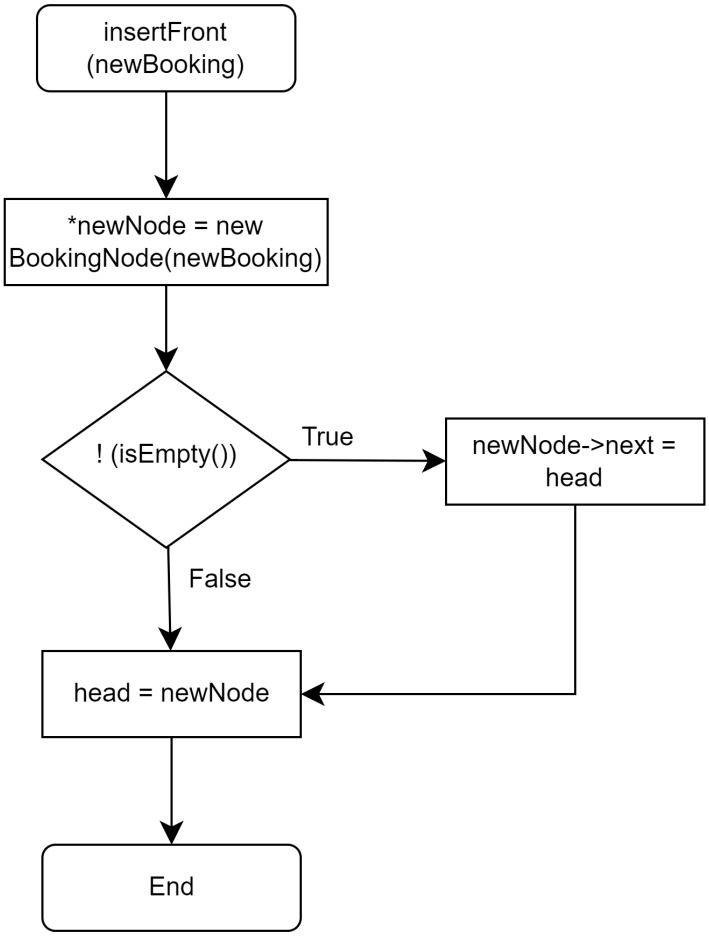

Figure 8: Flowchart of insertFront Function

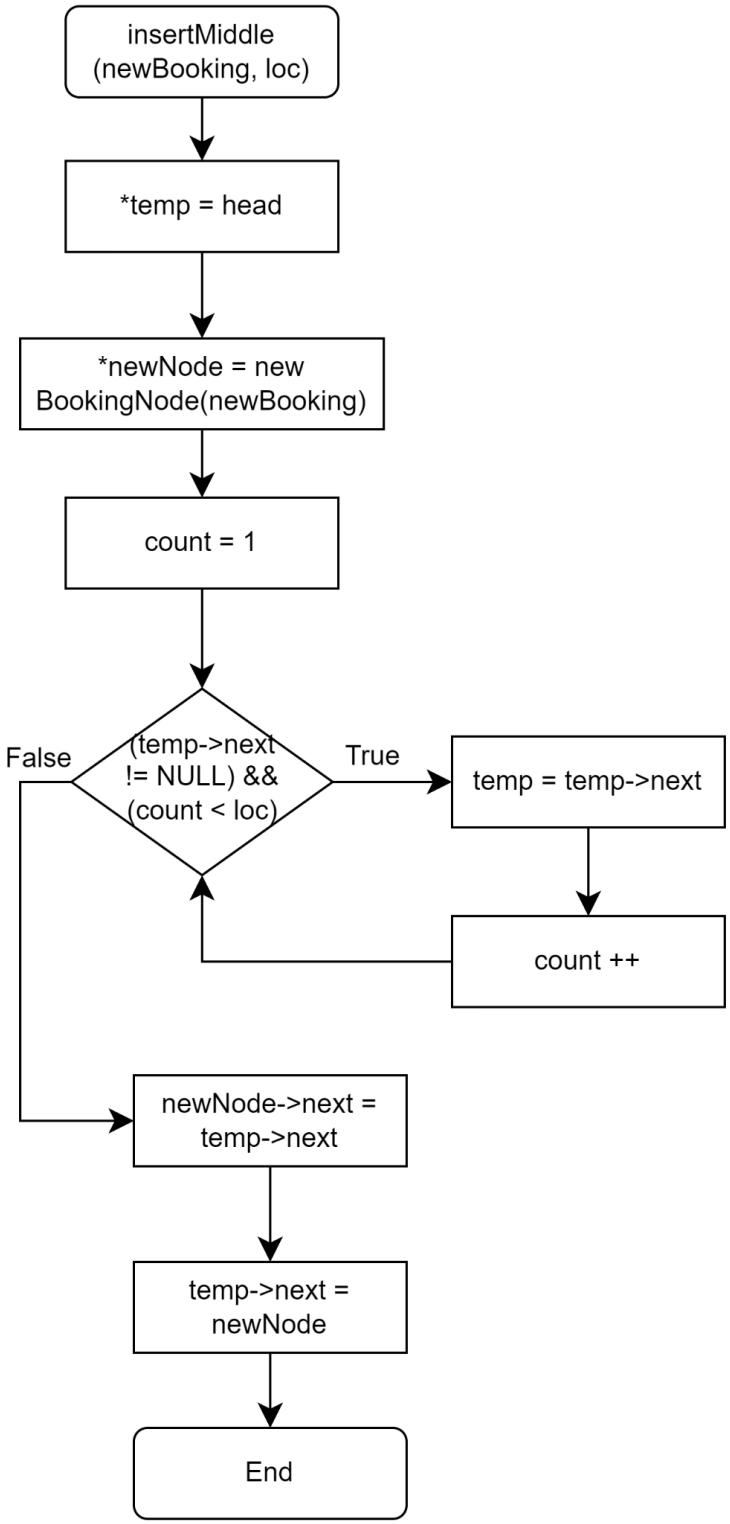

Figure 9: Flowchart of insertMiddle Function

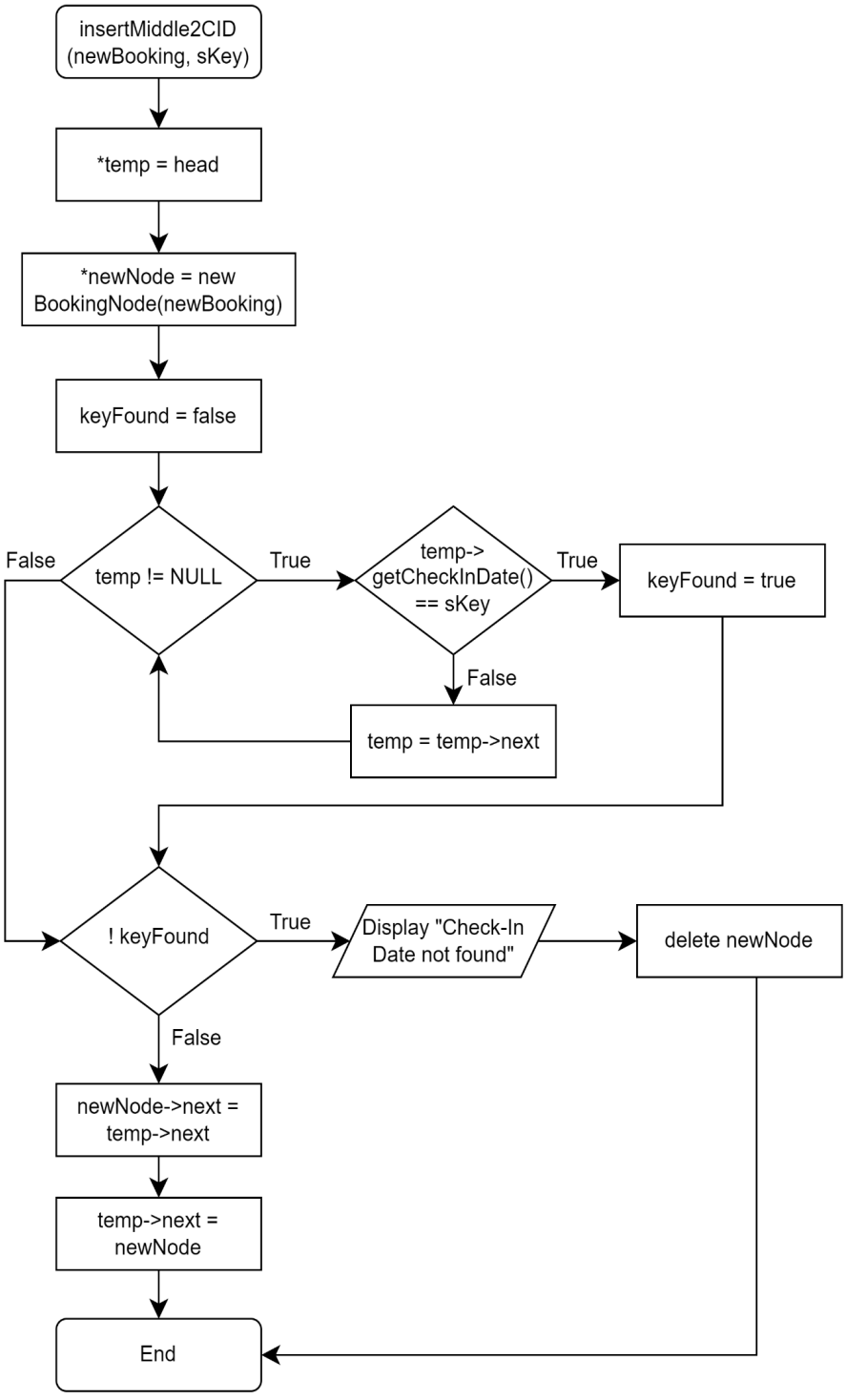

Figure 10: Flowchart of insertMiddle2CID Function

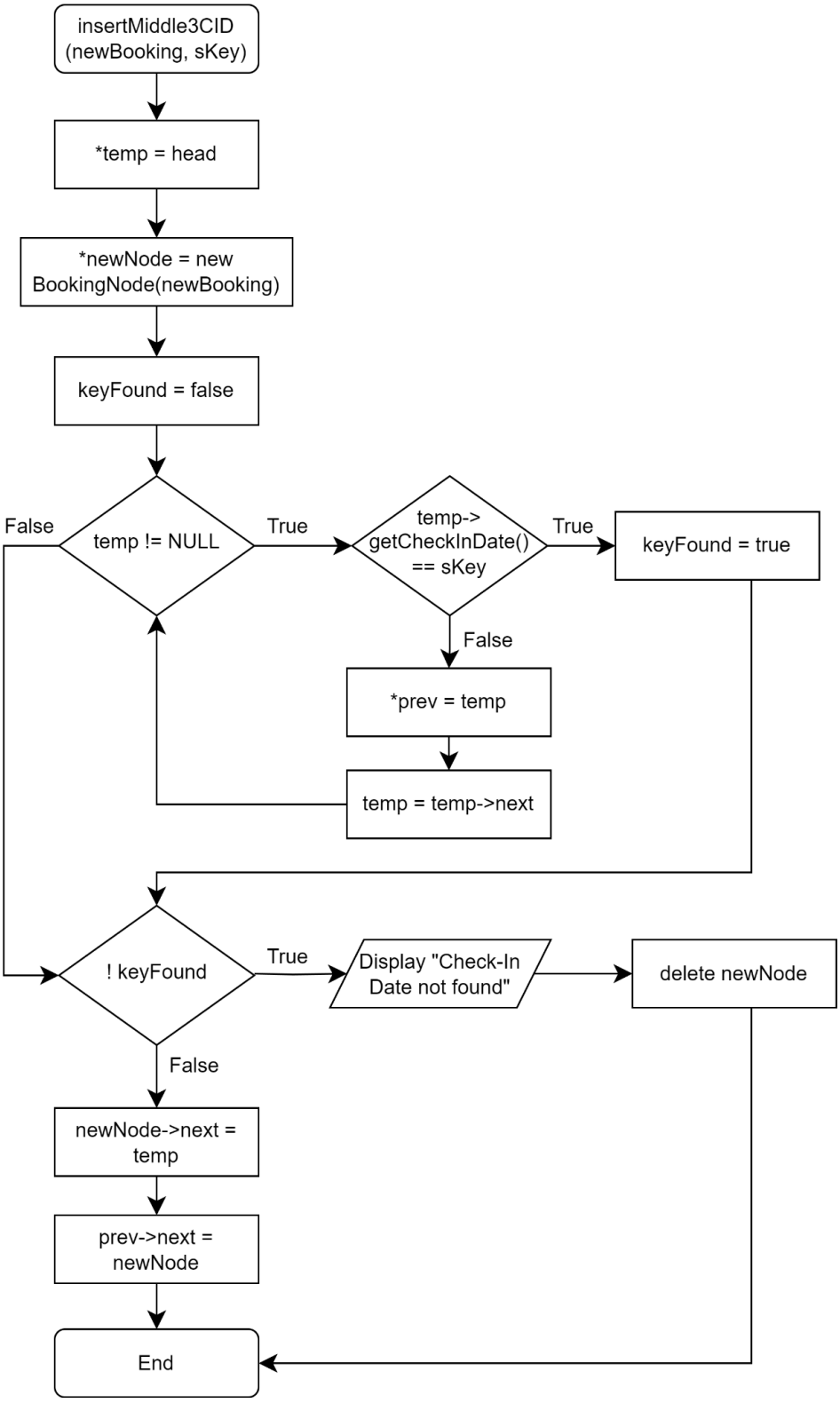

Figure 11: Flowchart of insertMiddle3CID Function

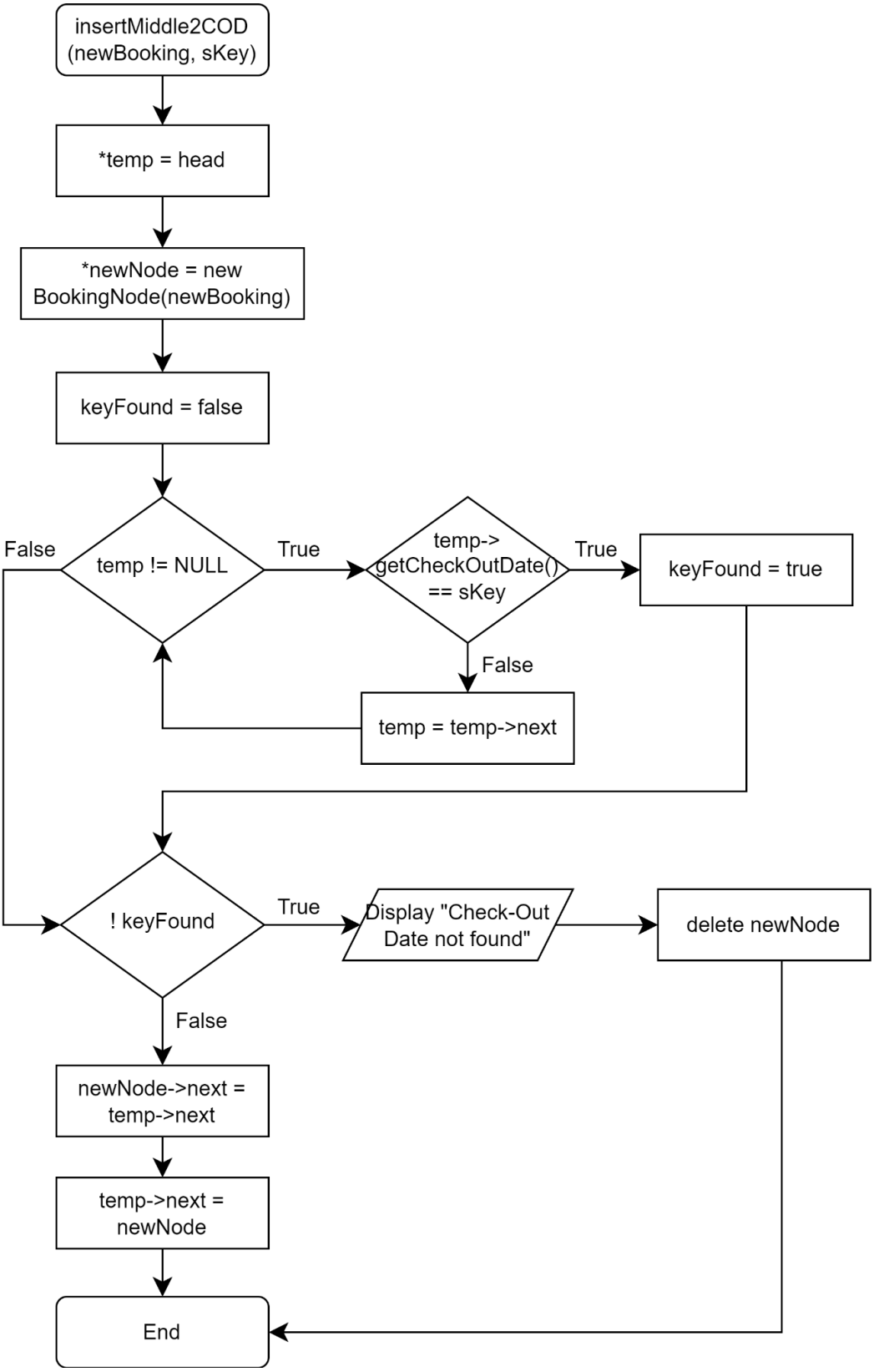

Figure 12: Flowchart of insertMiddle2COD Function

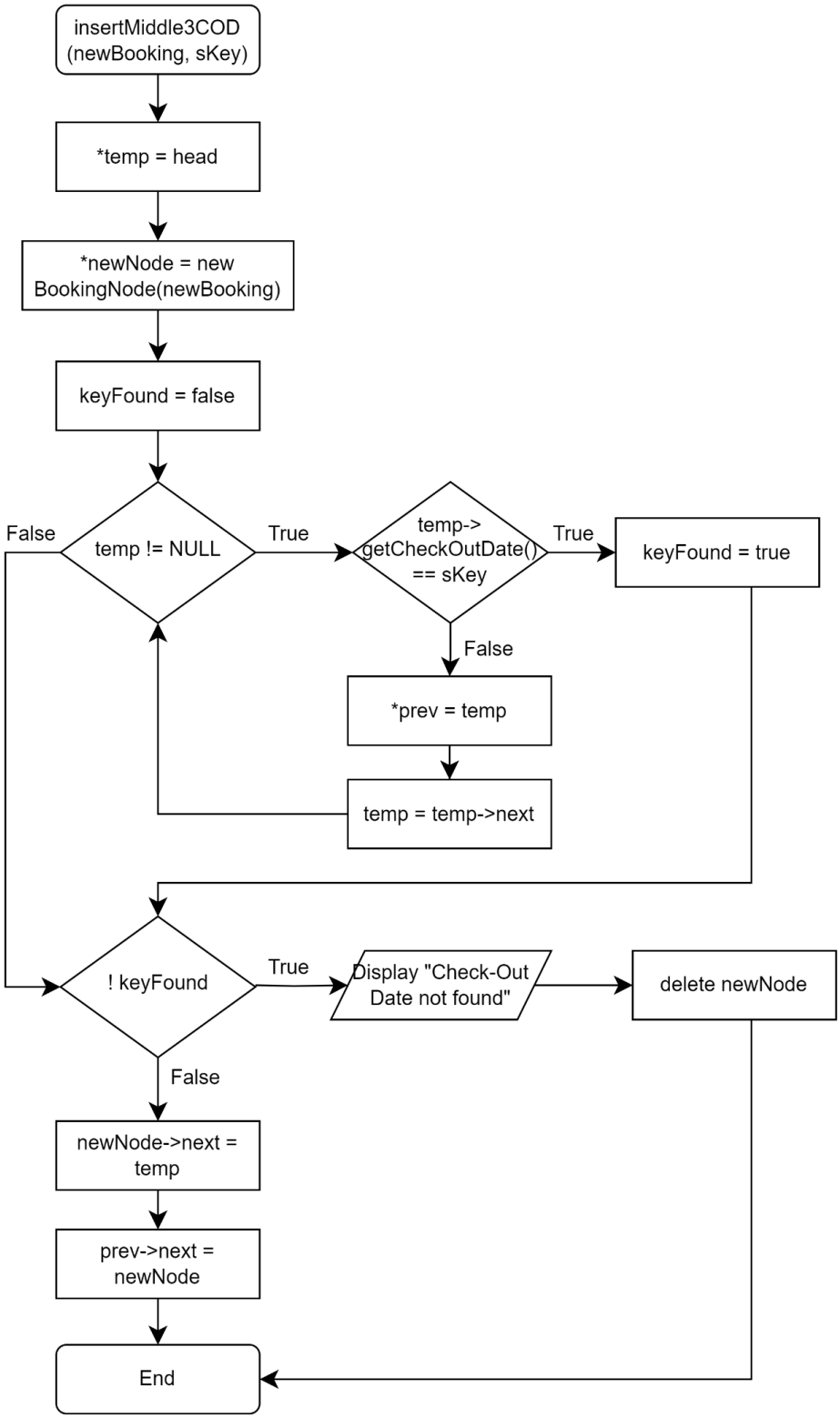

Figure 13: Flowchart of insertMiddle3COD Function

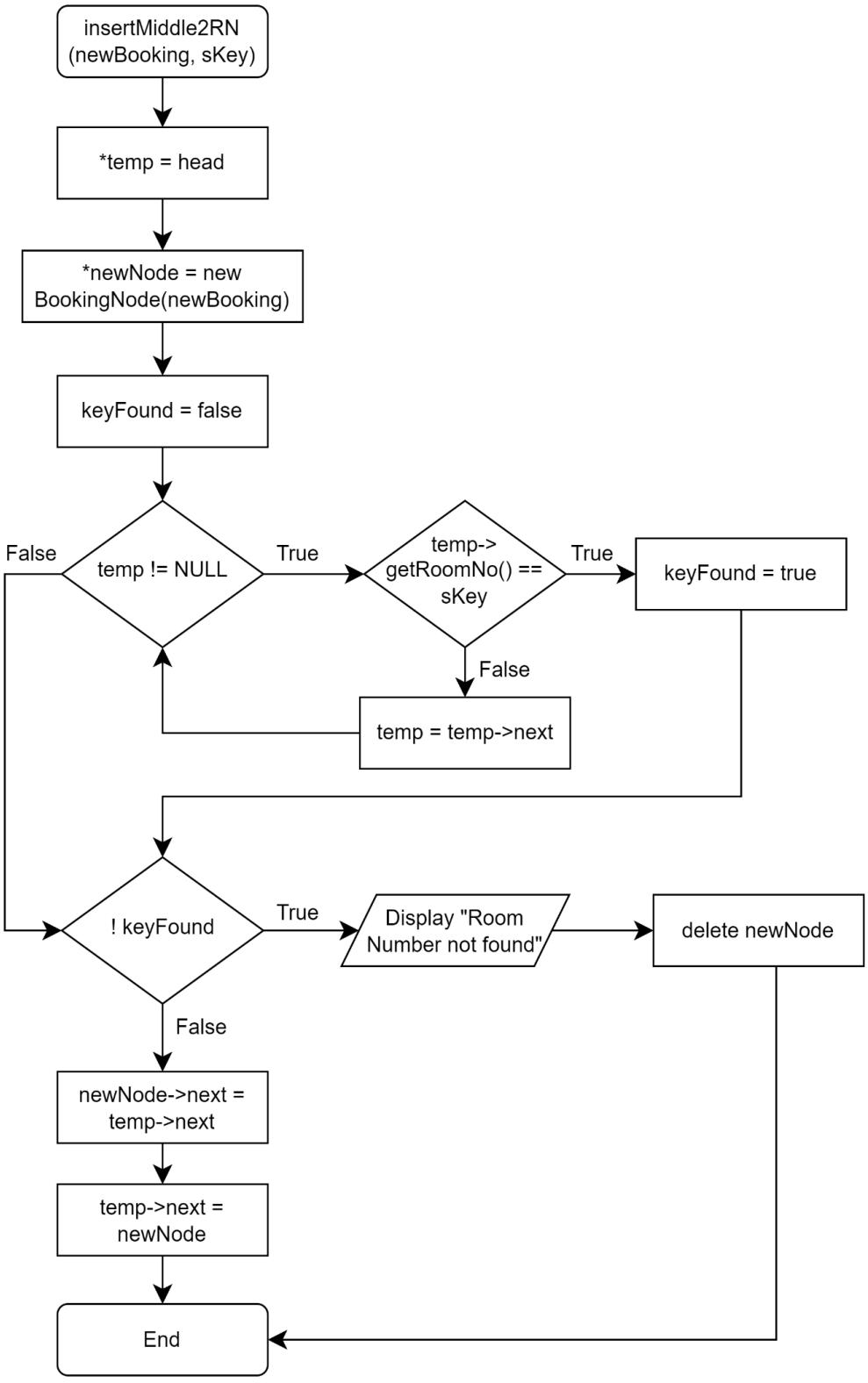

Figure 14: Flowchart of insertMiddle2RN Function

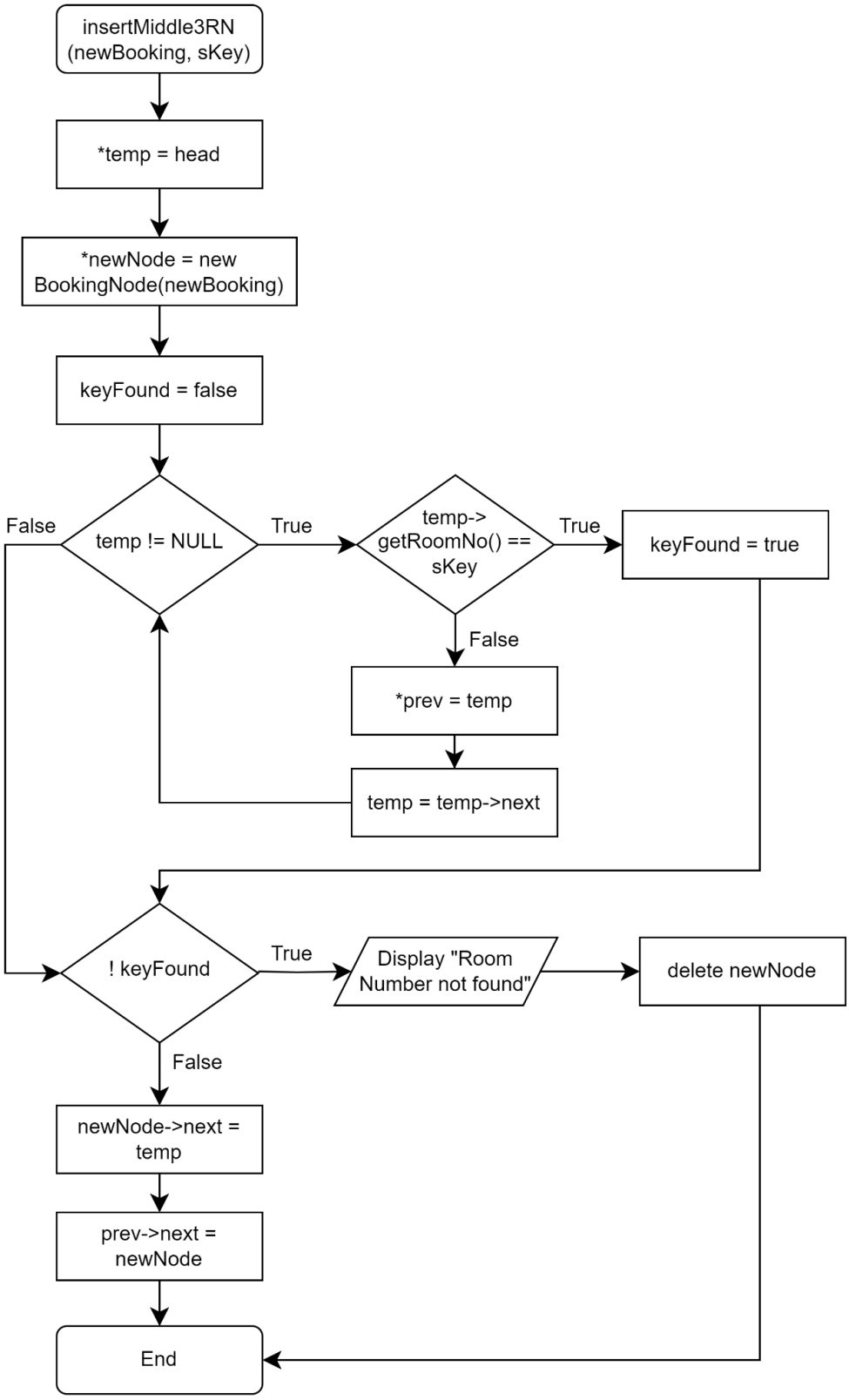

Figure 15: Flowchart of insertMiddle3RN Function

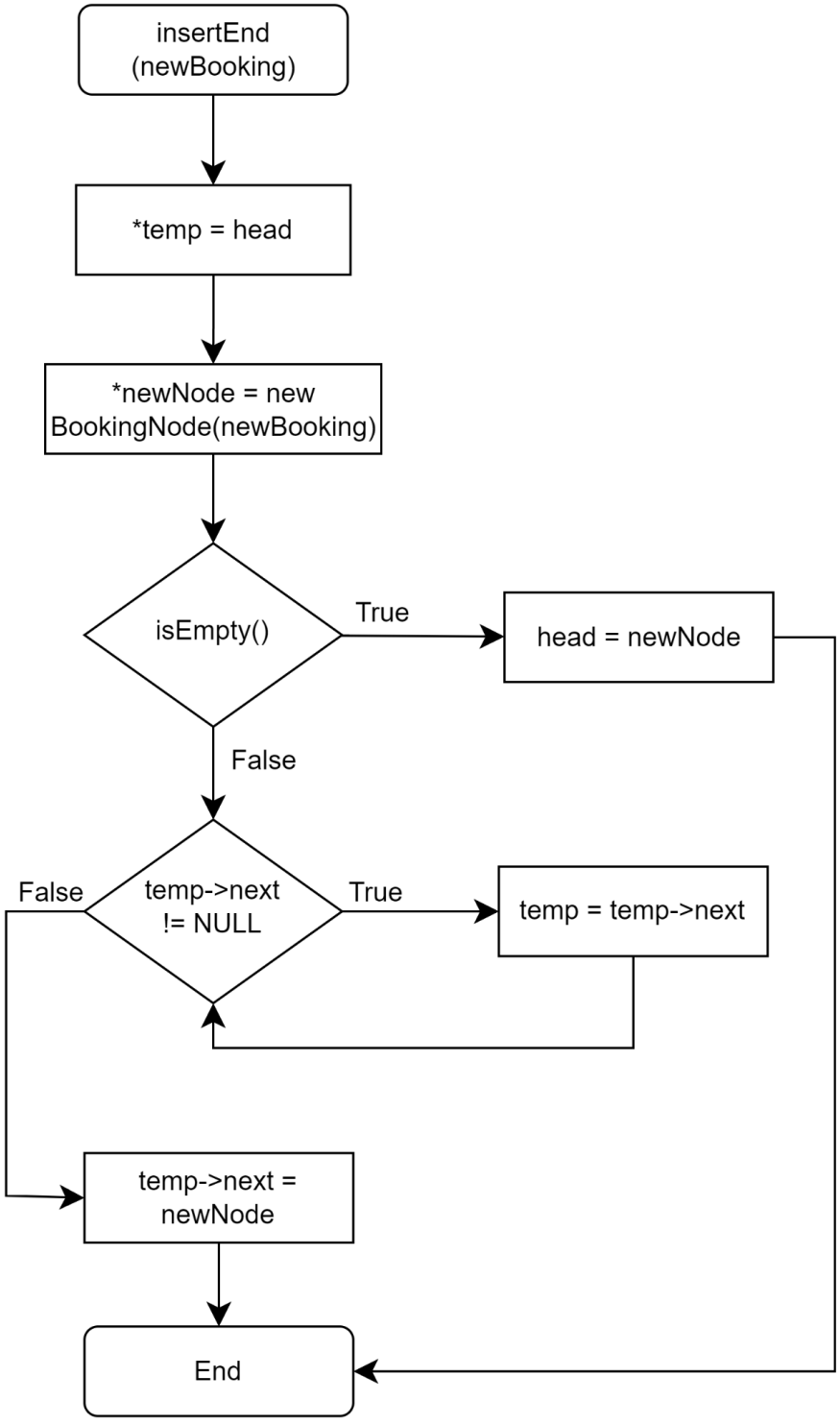

Figure 16: Flowchart of insertEnd Function

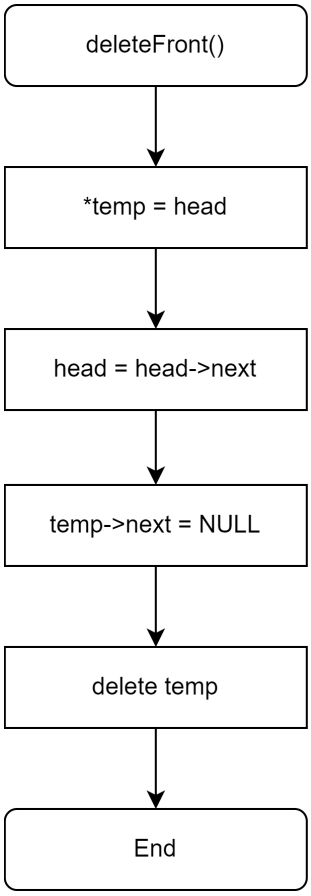

Figure 17: Flowchart of deleteFront Function

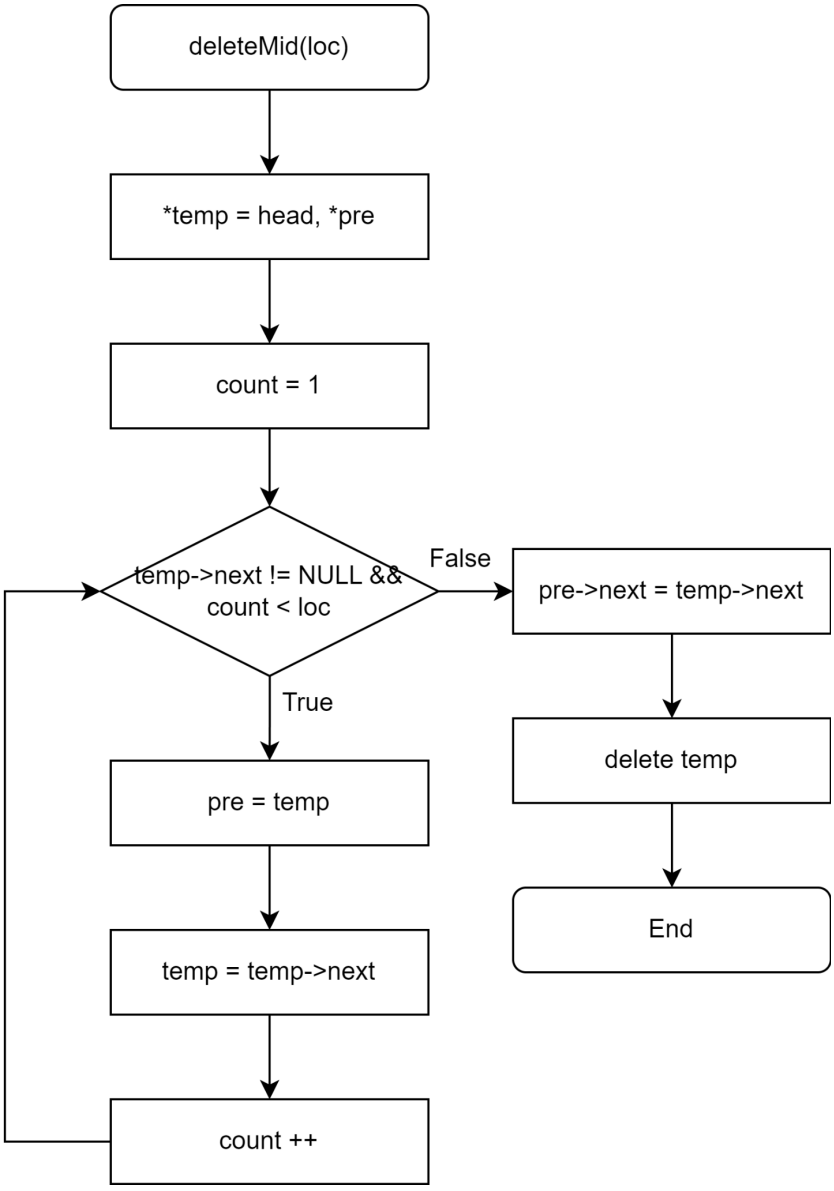

Figure 18: Flowchart of deleteMid Function

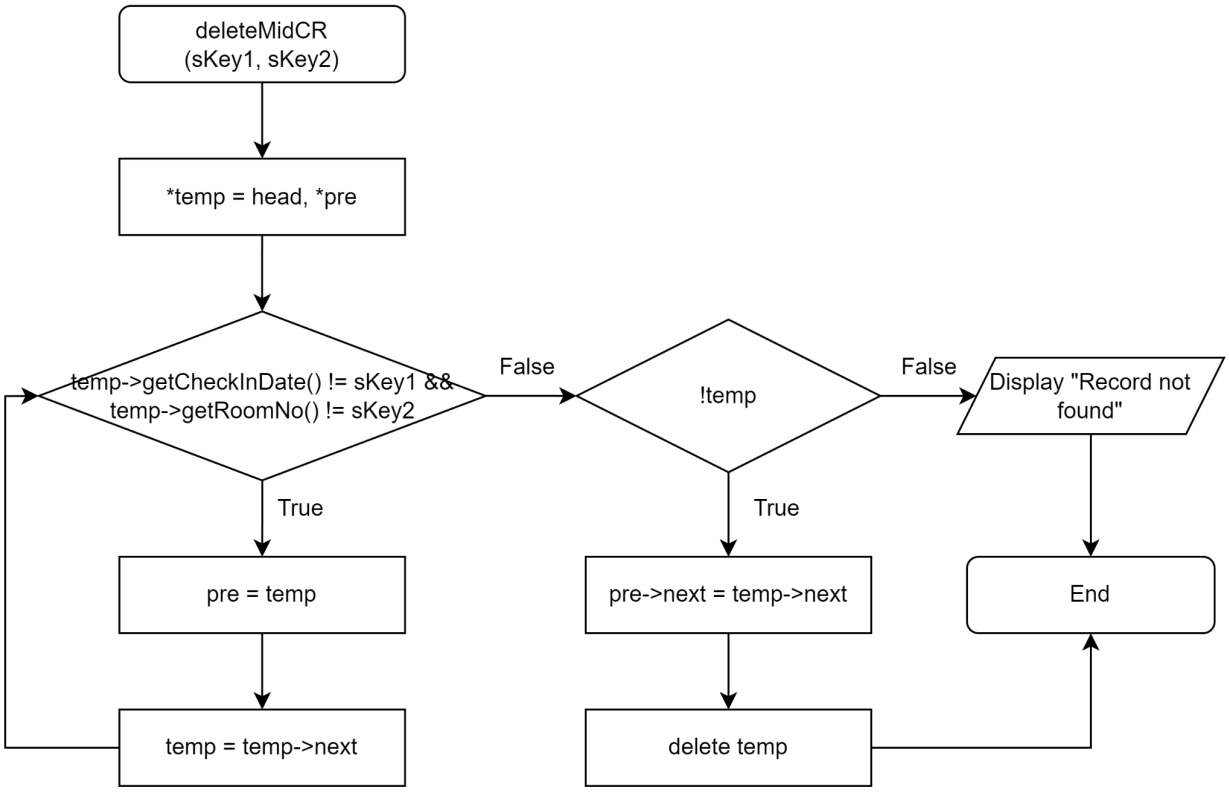

Figure 19: Flowchart of deleteMidCR Function

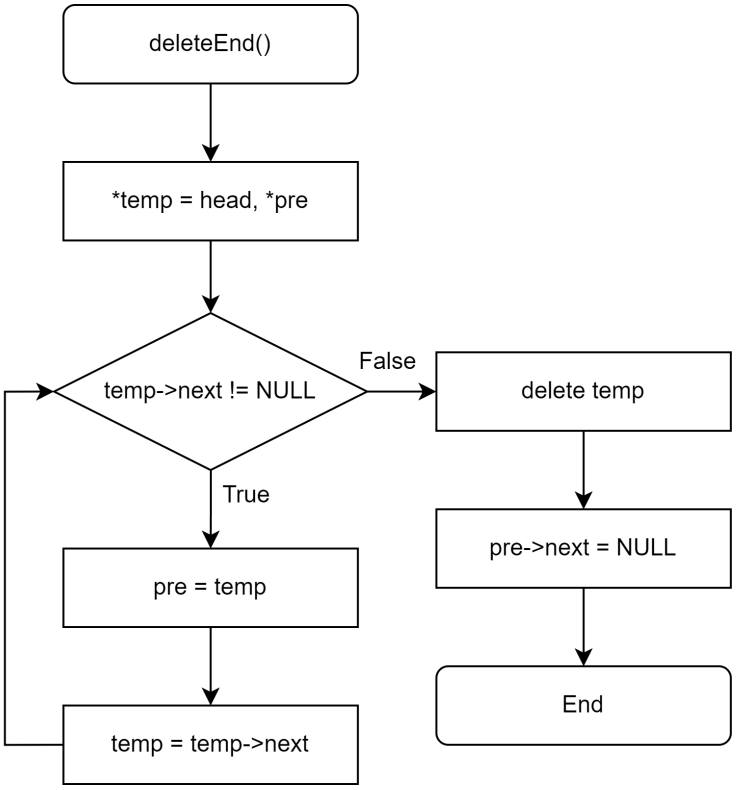

Figure 20: Flowchart of deleteEnd Function

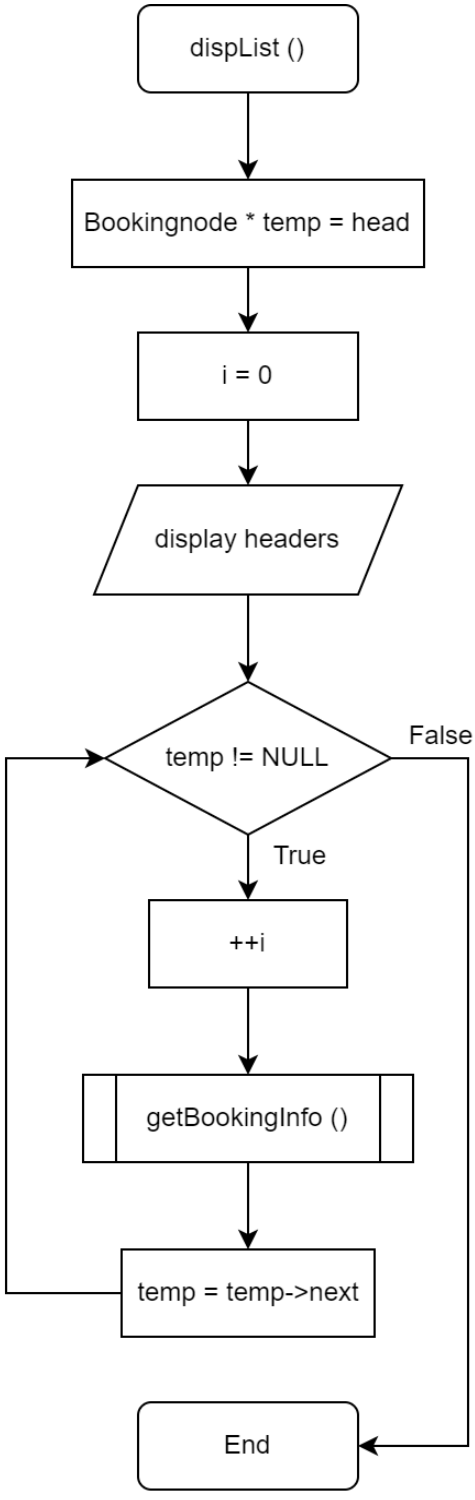

Figure 21: Flowchart of dispList Function

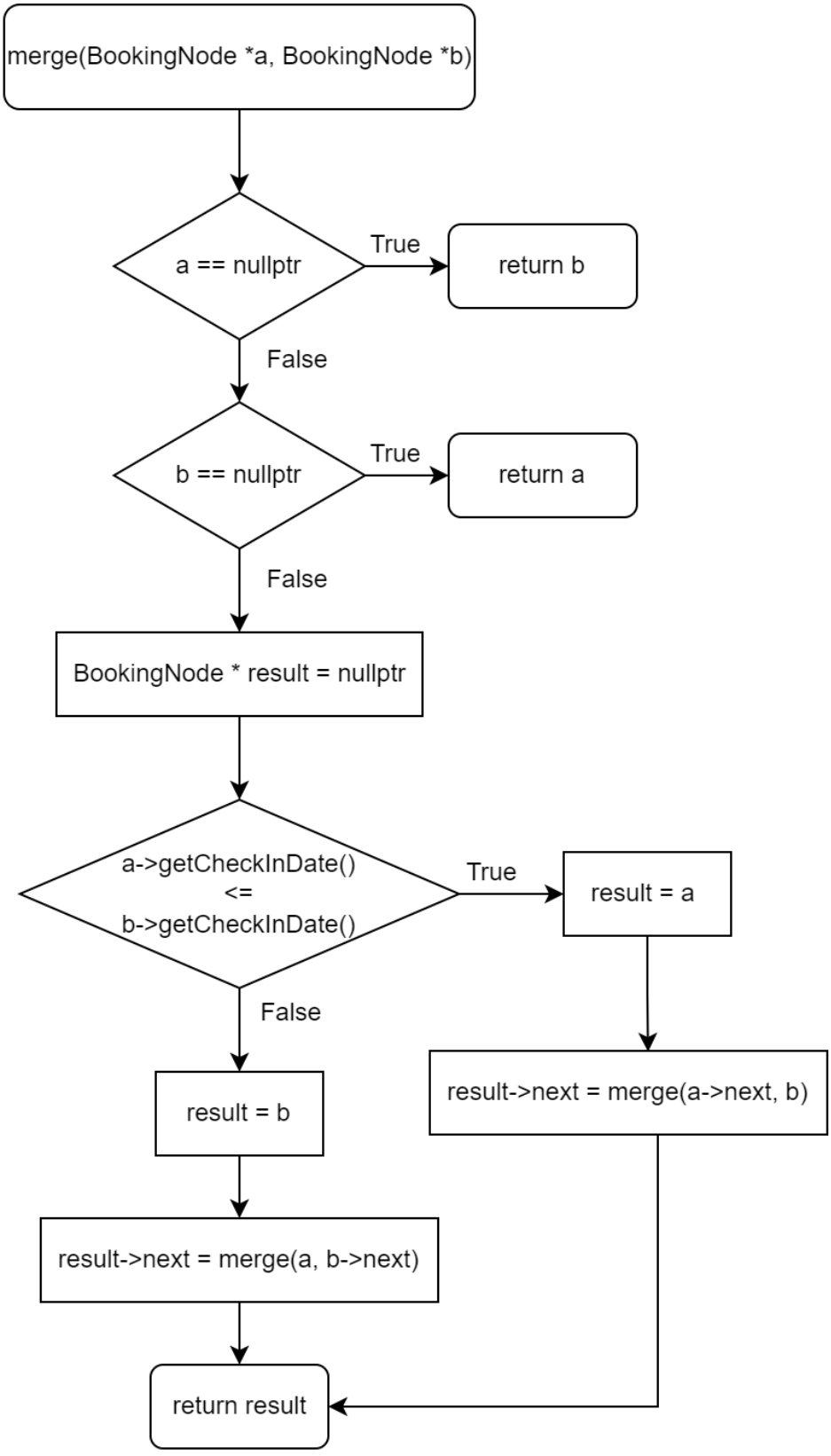

Figure 22: Flowchart of merge Function

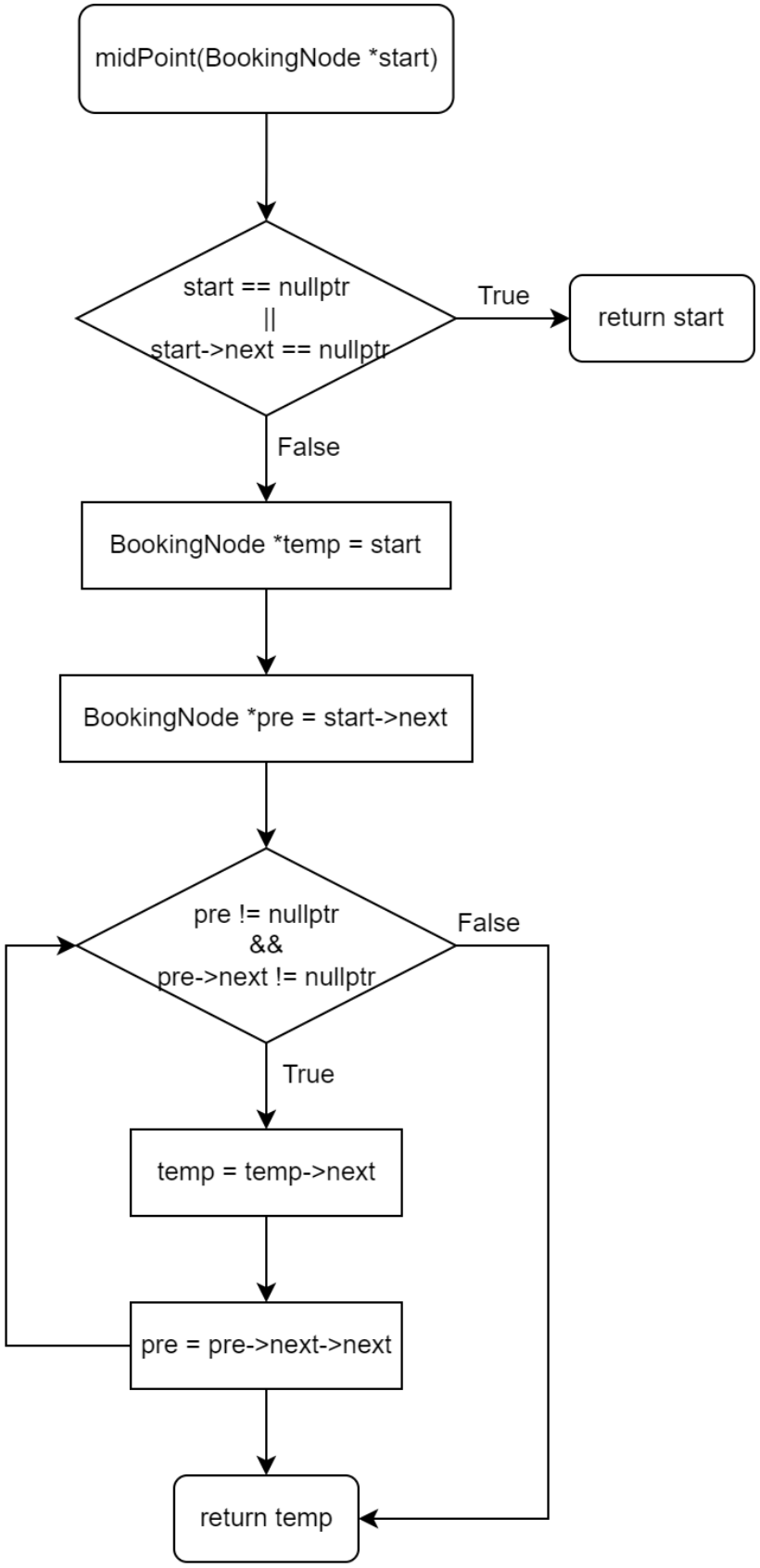

Figure 23: Flowchart of midPoint Function

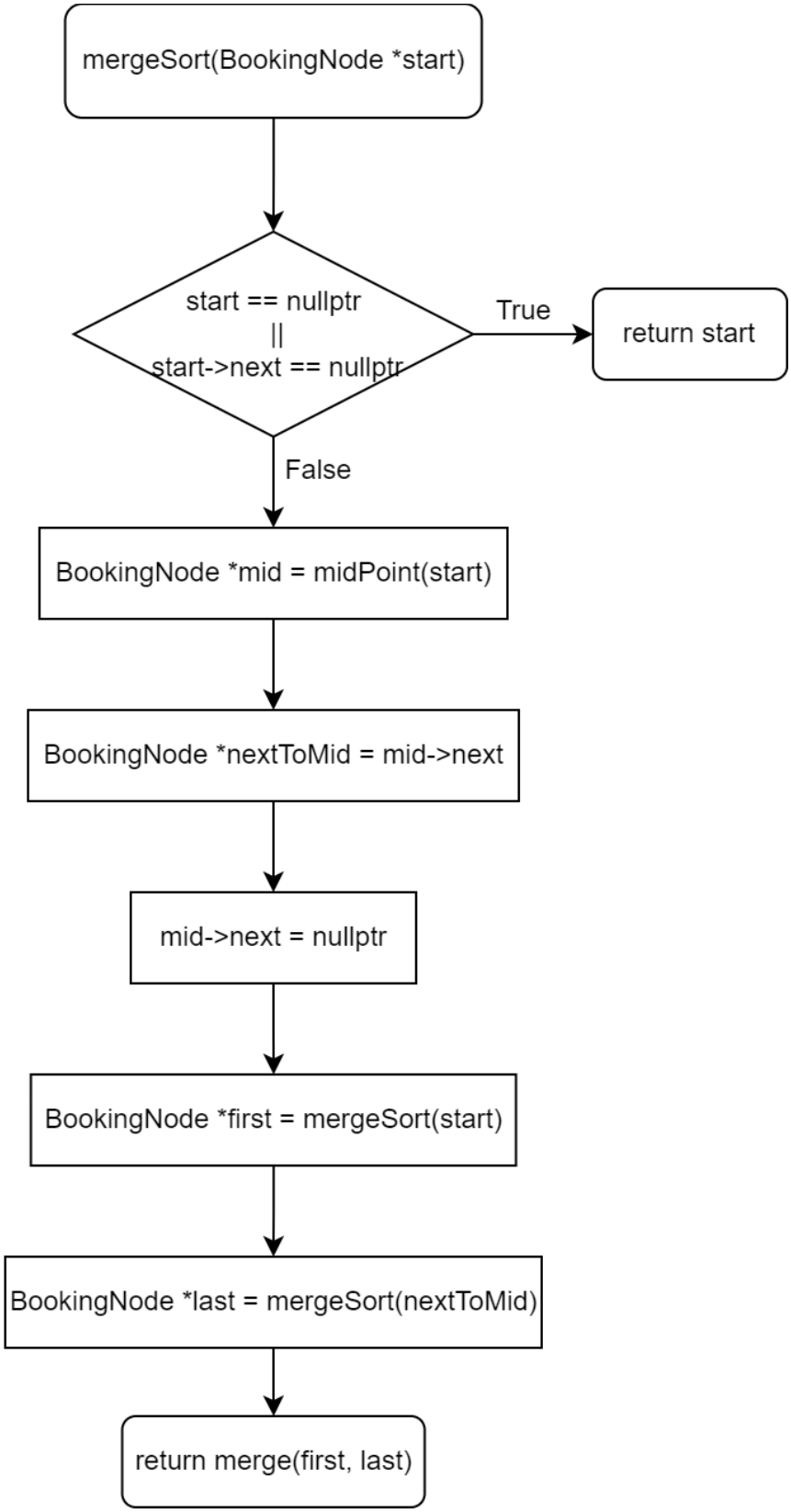

Figure 24: Flowchart of mergeSort Function

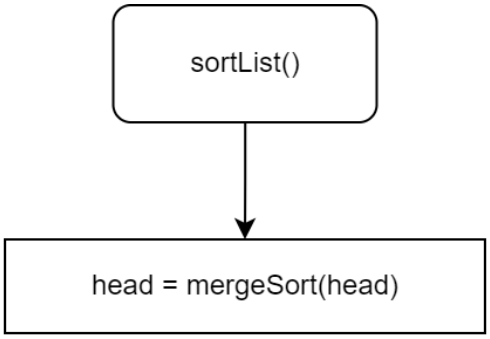

Figure 25: Flowchart of sortList Function

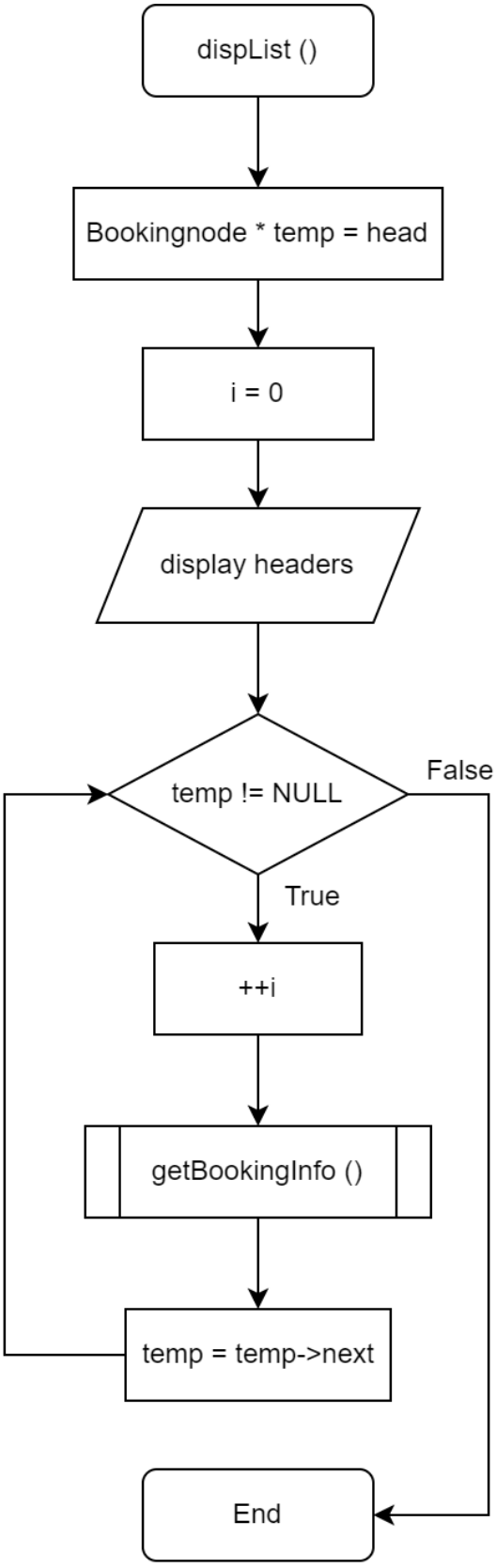

Figure 26: Flowchart of dispList Function

Description of how to implement data structure operation: Linked List

In the Hotel Booking System, we have implemented Linked List - Basic Operations - Inserting, Deleting, Finding, Displaying and Sorting in the form of Singly Linked List variation to define functions in managing the room bookings in a hotel. This data structure operation works by having two main classes which are used to declare node that contains data and a next pointer and list that contains a head pointer and the basic operations of a linked list.

1. **Inserting**

   There are four main types of insert linked list functions for inserting operation in our system, which are insert at the front, insert at the middle based on location, insert at the middle based on the data and insert at the end.

   First, insert at the front (insertFront()), which is implemented by checking if the linked list is empty first. If the linked list has any node, the new node will point to the first node of the linked list. Then, the head points to the new booking data. Besides, for insert at middle based on location (insertMiddle), a temporary pointer is declared and pointed to the first booking data at first. After that, the temporary pointer will continuously point to the next node until the location specified is reached, the new node will insert at the location by pointing the new node to the next node pointed by the temporary pointer and then the node pointed by the temporary pointer will be pointed to the new node. Furthermore, for inserting the node at middle based on specified data such as insertMiddle2CID(), insertMiddle3CID(), insertMiddle2COD(), insertMiddle3COD(), insertMiddle2RN() and insertMiddle3RN() in our system, it is similar to insert at middle based on location, but the data specified instead of location will be found using loop. After the data is found, the new node can be inserted after the data using the same method as insert based on location. However, for inserting before the data, another pointer (*pre) will be declared in addition to temporary pointers. It will point to the node before the node pointed by the temporary pointer. Once the data is found, the next pointer of the new node will point to the node pointed by the temporary pointer and the next pointer of the node pointed by the pre pointer will point to the new node to complete the insertion

operation. Lastly, the insert at the end (insertEnd()) function is implemented by declaring the temporary pointer and pointing to the first booking data of the linked list. The linked list will be checked if it is empty. If it is empty, the head will directly point to the new node. If the linked list is not empty, the temporary pointer will be moved to point to the next booking data until the next pointer of that node points to NULL. Then, the new booking data will be inserted after that node.

2. Deleting

In the Basic Operations - Deleting, we have defined four delete linked list concept functions which are deleteFront(), deleteMid(), deleteMidCR() and deleteEnd() functions. The delete operations are implemented by deciding and locating the desirable data to be deleted, setting the pointer of the predecessor of the unwanted data to the successor of the data, declaring the pointer of data that points to the successor to NULL (for deleting first data) and releasing the memory used by the data.

As an example, in our system, the deleteFront() function is used to delete the first booking data by declaring a temporary pointer points to head (first data), pointing head to the data pointed by the next pointer of head to make the second booking data as the first booking data and breaking the pointer connection by declaring the next pointer of the temporary pointer to NULL and finally deleting the temporary pointer. For deleteMid() function, the desirable unwanted booking data location is entered and found using a while loop, points the next pointer of the previous booking data pointer to the booking data pointed by the next pointer of unwanted data and ultimately deleted the booking data found while the deleteMidCR() function will be implemented in a similar way except the function will accept, find and delete the booking data that has matched check-in date and room number at the same time. Lastly, the deleteEnd() function operates by finding and deleting the last booking data and making the second last booking data points to NULL.

3. **Finding**

The finding operation is implemented by utilizing a counter to track node positions and matching the user input to every node value. Once it finds the match, the position value will be returned to help the user identify the booking. In our case, the findNode() function is used to locate a customer's IC in a booking list.

First, the findNode() function takes user input as the string parameter. A temporary pointer is set to the first node of the list, and a counter is initialized to 1. Then, a while loop runs as long as the temporary pointer is not null and the IC of the node does not match the user input. Inside the loop, the temporary pointer points to the next node, and the counter is incremented. Once the match is found or the pointer is null, the loop stops. Then the function checks the temporary pointer. It will return the position if the pointer is null, otherwise it displays an error message.

4. **Displaying**

The dispList() function and getBookingInfo() function are used to display the list of customer bookings which contains check-in date, check-out date, room number, room type, customer IC, and total price.

In dispList() function, a temporary pointer is created and pointed to the first node of the list. A counter is also initialised to 0. The while loop will check whether the temporary pointer is null. If not, it pre-increments the counter, points the pointer to the next node and calls getBookingInfo() function. The getBookingInfo() function is used to display booking details for each node.

5. Sorting

We have implemented Merge Sort algorithm to sort bookings ascendingly based on check-in date using sortList(), mergeSort(), midPoint() and merge() functions. By dividing the linked list recursively into two lists after finding the midpoint, two lists will be compared and sorted then merged back into the original list.

First, sortList() function is called in the main function. The result of mergeSort() function is declared as the first node of the list. mergeSort() function will then check whether the start node of the list is empty. If so, it returns the start node. If not, midPoint() function is called to get the midpoint. The list is then separated into two sublists using the midpoint. The sublists will be separated recursively until all lists with one or zero nodes are left and return the result of merge() function by taking two sublists as parameters.

Inside merge() function, it checks whether one list is empty. If not, it compares the two nodes by their check in date. A new list is created and the node with earlier check in date is inserted as the first node. The merging process will ensure all nodes are merged and return the new list.