DATA STRUCTURES

Table of Contents

Group Members:

NAME	**TP NUMBER**
ARSHAD KHALID NAZIR	TP058473
MARI RAIMBEKOVA	TP059928
RAJA ZARIFAH DINA MOHAR BINTI RAJA MAZHAR MOHAR	TP059868

Section 1: Introduction

Q-OnIT is a Malaysian software development firm that creates, markets, and deploys software solutions to solve specific client requirements. Arshad Khalid, Mari Raimbekova, and Raja Zarifah make up the company's software development team.

Klinik Sulaiman has approached Q-OnIT with a challenge that required to be addressed. The firm was tasked with developing a patient queue management system for the clinic, with two user roles in mind: doctor and nurse.

As the software development team, we will build the program using the C++ programming language and incorporate two linked lists for this venture.

Section 2: Data Structures

Linked Lists

A linked list is a form of linear data structure in which all the components are not always embedded in consecutive storage addresses. A linked list is a collection of nodes that contain data and/or links. The link serves as a pointer towards the following link (Data Structure and Algorithms - Linked List, 2021). There has to be a beginning inside a list. As a result, we assign the very first node's address an identifier: "head." The final node in the linked list can also be acknowledged since it references "NULL."

Representation

The illustration below depicts how linked lists are composed of. i.e., A connection of nodes that hold data and links (next), with a beginning node named "Head" and the last node pointing to "NULL" to signify the end of the list.

Figure 1; Linked List Representation (Source: Tutorialspoint.com)

Types of Linked Lists

Singly Linked List

The most frequently used kind of lists are singly linked lists. Each node here is comprised up of data and "next" (that points to the next node). Node traversal in singly linked lists is solely unidirectional (Linked List Data Structure, 2021).

Figure 2; Singly Linked List (Source: Programiz.com)

Image below how a node in a singly linked list is represented:

```
struct node
{
  int data;
  struct node *next;
};
```

Figure 3; Linked list representation

Complexity

Time Complexity						Space Complexity
Average			Worst			Worst
Search	Insert	Delete	Search	Insert	Delete	
0(n)	0(1)	0(1)	0(n)	0(1)	0(1)	0(n)

Source: https://www.bigocheatsheet.com/

Doubly Linked List

Doubly linked lists are another variety of linked list. In this instance, node direction might be both forwards and backwards. There is a "previous" that links to the previous node, a "next," as well as the data in each node.

Figure 4; Doubly Linked List (Source; Programiz.com)

Image below how a node in a doubly linked list is represented:

```
struct node {
    int data;
    struct node *next;
    struct node *prev;
}
```

Figure 5; Doubly linked list representation

Complexity

Time Complexity						Space Complexity
Average			Worst			Worst
Search	Insert	Delete	Search	Insert	Delete	
0(n)	0(1)	0(1)	0(n)	0(1)	0(1)	0(n)

Source: https://www.bigocheatsheet.com/

Circular Linked Lists

A circular linked list is one in which the final node points to the very first, establishing a circular loop. In this circumstance, a circular list can either be singly or doubly linked. When the list is singly linked, the "next" of the final node links to the first node, whereas in a doubly linked list, the "prev" of the first node now links to the last node. This implies that within a circular linked list, there is no "NULL".

Comparing the linked lists

Comparison	Singly Linked List	Doubly Linked List
Efficiency	It's much less efficient than a doubly linked list as direction is forwards.	It is more efficient as direction is in both directions.
Memory Consumption	Less memory consumption as each node consists of just data and one pointer	Doubly linked list uses more memory as there is data and two pointers per node
Usage	Can be implemented on the stack.	It is possible to integrate it on a stack, heap, or binary tree.
Complexity	The time complexity of insertion and deletion of a node at a known position is O(n).	The time complexity for inserting and deleting a node is O(1).
Index performance	Singly linked list is preferred when we need to save memory and searching is not required as pointer of single index is stored.	If we need better performance while searching and memory is not a limitation in this case doubly linked list is more preferred.

Note. The table of comparison are from "Difference between Singly linked list and Doubly linked list in Java" by tutorialspoint, 2021. Difference between Singly linked list and doubly linked list in Java. (2019). Tutorialspoint.com. https://www.tutorialspoint.com/difference-between-singly-linked-list-and-doubly-linked-list-in-java

Justification

As seen in the comparison table above, a singly linked list is exceedingly simple to implement and necessitates very little memory to be utilized for storage than its counterpart. The tradeoff would be that, because sequence is one-way, we must keep a handle towards the initial node or else the list would vanish. In order to discard a node, you have to start at the beginning of the list and traverse through it until you locate the node prior to the one that is being deleted. Doing so takes O(n) time, thus the worst-case time complexity for the removal process is O(n).

In contrast to a doubly linked list, there is far less risk of data loss since the list may be recapitulated back and forth, improving performance. Nevertheless, this would be difficult to implement and demands considerably increased capacity for storage. Removal and insertion of nodes are substantially less time consuming, because to erase a node when a pointer to that node is specified, a doubly linked list usually requires O(1) under the worst case. Due to its inherent efficiency, doubly linked lists are best employed whenever searching is a primary priority. For these reasons, we have decided that a doubly linked list will be used.

Proposed Source Code

The declared function preDefinedRecords() consists of data of 5 patients as no external storage is allowed. Hence data will be recorded and run before the whole system is turned on.

```
createPredefinedRecords();
```

Figure 6; createPreDefinedRecords() function

Function below prompts the nurse to enter their credentials to access their respective menu.

```
nurseLogin();
```

Figure 7; nurseLogin() function

Function below prompts the doctor to enter their credentials to access their respective menu.

```
doctorLogin();
```

Figure 8; doctorLogin() function

The nurse() function displays the menu available to the nurse.

```
void nurse();
```

Figure 9; nurse() function

The doctor() function displays the menu available to the doctor.

```
doctor();
```

Figure 10; doctor() function

Function below adds a new patient to the waiting list by the nurse.

```
void addPatientWaitingList();
```

Figure 11; addPatientWaitingList() function

Function declared to allow the nurse to change the patient order in the waiting list.

```
changePatientOrder();
```

Figure 12; changePatientOrder() function

Function to view the original list of the patients by either the doctor or nurse

```
void viewPatients();
```

Figure 13; viewPatients() function

Function that automatically transfers the patient from the waiting list to the history list as they go to see the doctor.

```
void callPatient();
```

Figure 14; callPatient() function

Function that allows for the search of a patient in the waiting list by their patient ID.

```
searchPatientFromWaitingListByPatientID();
```

Figure 15; searchPatientFromWaitingListByPatientID() function

Function to sort and display the patients from the waiting list according to the visit time arranged.

```
void sortPatientbyVisitTimeAscending();
```

Figure 16; sortPatientbyVisitTimeAscending() function

Function that allows the doctor to search a specific patient from the visit history and have the ability to modify their records.

```
searchAndModify();
```

Figure 17; searchAndModify() function

The function to sort and display all the patients in the visit history in descending order using their first names.

```
sortAnddisplayPatientHistoryFnameDescending();
```

Figure 18; sortAndDisplayPatientHistoryFnameDescending() function

Function that is able to search a patient from the patients visit history based on sickness description or their first name.

```
void searchPatientHistorybySicknessOrFname();
```

Figure 19; searchPatientHistorybySicknessOrFname() function

Function below displays lists in a page-by-page view

```
displayPagebyPage();
```

Figure 20; displayPagebyPage() function

Function below moves page forwards

```
nextPage();
```

Figure 21; nextPage() function

Function below moves page backwards

```
previousPage();
```

Figure 22; previousPage() function

Below function searches a patient using their patient ID

```
searchPatientID;
```

Figure 23; searchPatientID() function

Below function searches patient by their First names or sickness.

```
generalSearchPatient(searchType, search, head);
```

Figure 24; generalSearchPatient(searchType, search, header) function

Below function adds a patient to the end of a temporary list

```
addPatientEndOfTempList();
```

Figure 25; addPatientEndOfTempList() function

Function to delete a temporary list

```
deleteTempList();
```

Figure 26; deleteTempList() function

Function to display out a patient details whenever the function is called

```
displayPatientDetails();
```

Figure 27; displayPatientDetails() function

Function created a temporary list to store patient details

```
createTempPatientDetails();
```

Figure 28; createTempPatientDetails() function

The patient structure that consists of all the information that makes up a patient's detail needed by the clinic.

```
struct patient
{
    string PatientID;
    string Fname;
    string Lname;
    string gender;
    int age;
    bool disability;
    int phoneNumber;
    string address;
    string sickness;
    string VisitDate;
    string VisitTime;
    string DoctorName;
    bool status;
    patient* nextaddress;
    patient* prevaddress;
} *newnode, * current, * patientHead, * patientTail,
* historyHead, * historyTail, * tempHead, * tempTail, temporary;
```

Figure 29; patient struct

A function to add a new patient node that contains the patients details at the end of the list

```
void createAnInsertnewNodeToEnd(string PatientID, string Fname, string Lname,
    string gender, int age, bool disability, int phoneNumber, string address,
    string sickness, string VisitDate, string VisitTime, string DoctorName, bool status);
```

Figure 30; function to add new patient to list

Visualization

Below are the doubly linked lists for both the patients waiting list and history list. As we can see, the data is capable of moving forwards and backwards without the need of a loop with the use of the head, tail and current (which is not shown currently).

Figure 31; Patient waiting list doubly linked list

Figure 32; Patient history list doubly linked list

Section 3: Workflow Planning

main()

Algorithm

1. Create the menu for the system user.
2. Call the createPredefinedRecord() function.
3. Doctor's password is a "clinicDoctor"
4. Nurse's password is a "clinicNurse"
5. Display the options for the user.
6. Proceed with the user's option.
7. Use a switch function.
8. In case, the user chooses "1. Nurse Login", program will call nurseLogin() function and takes the user to the main menu under nurse account.
9. In other case if the user chooses "2. Doctor Login", program will call doctorLogin() function and takes the user to the main menu under doctor account.
10. In third case, where user chooses "3. Exit Program", program shuts down.
11. Else, by default, the program prints "Invalid Option" and takes the user to main menu to make a choice again.

Flowchart

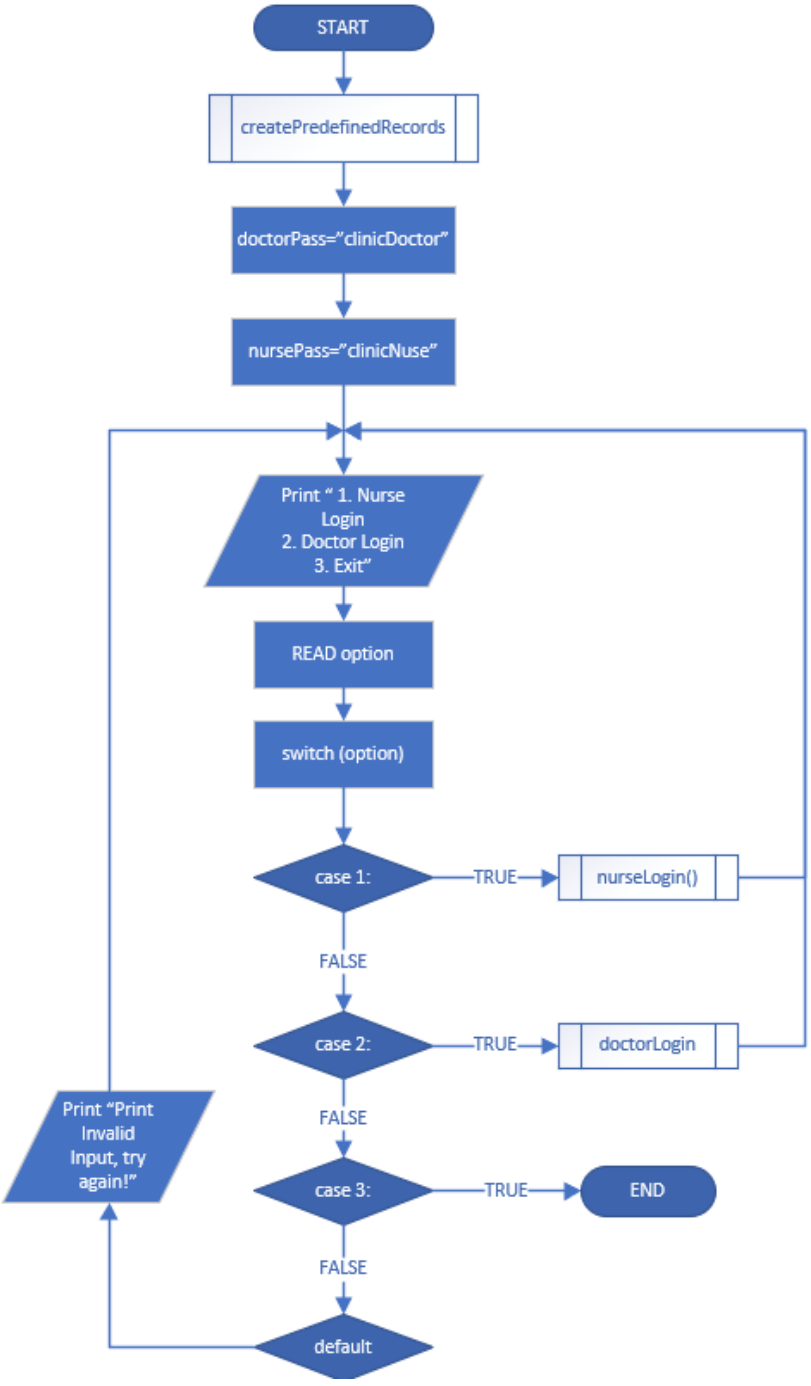

Figure 33; Flowchart of main () function

Brief Explanation

Given figure above demonstrates the flowchart for the main() function, where user has few options to log in into the system. First option is a “Nurse Login”, by choosing this program will call nurseLogin() function and user enters the system under the nurse account. Second option is a “Doctor Login”, where the program call doctorLogin() function and the user enters the system under the doctor account. Last option is to exit the program, by choosing this user shuts down the program. In case, none of the options were entered or wrong option was entered then program prints “Invalid option” by default and takes user to the main menu.

createPredefinedRecords()

Algorithm

1. Create predefined records.
2. Global variable that includes newnode, current, patientHead, patientTail, historyHead, historyTail, tempHead, tempTail, null.
3. Call the following function to pass: "P001", "Dorian", "Grey", "Male", 29, "False", 011990088, "9, Jalan Cemur, 50400, KL", "Anemia", "12/11/2021", "10.00", Dr. Low, True.
4. Call the following function to pass: "P002", "Ali", "Khan", "Male", 21, "True", 012440066, "3, Jalan Klang, 50403, Selangor", "Diabetes", "13/11/2021", "15.00", Dr. Sofia, True.
5. Call the following function to pass: "P003", "Bayan", "Rasul", "Female", 35, "False", 014899066, "4, Jalan Ampang, 50422, Selangor", "Alzheimer's disease", "13/11/2021", "11.00", Dr. Sofia, True.
6. Call the following function to pass: "P004", "Lee", "Dickson", "Female", 21, "True", 014526456, "17, Jalan Bukit, 50600, KL", "Cancer", "13/11/2021", "12.30", Dr. Sofia, True.
7. Call the following function to pass: "P005", "Jason", "Bourne", "Male", 44, "False", 013599456, "12, Bukit Bintang, 50900, KL", "Checkup", "15/12/2021", "17.30", Dr. Low, True.
8. Call the following function to pass: "P006", "Yong", "Chong", "Female", 33, "False", 013532245, "11, Batu, 51400, KL", "Dengue", "4/02/2020", "14.00", Dr. Sofia, False.
9. Call the following function to pass: "P007", "Nadia", "Aleem", "Female", 22, "True", 019489698, "4, Jalan Bandar, 50660, KL", "Influenza", "27/07/2021", "12.30", Dr. Low, False.
10. Call the following function to pass: "P008", "Ali", "Mohsin", "Male", 12, "False", 015447895, "8, Sunway, 52500, KL", "Diarrhea", "15/09/2020", "16.15", Dr. Low, False
11. Call the following function to pass: "P009", "Liya", "Gonsalves", "Female", 52, "True", 013665492, "10, Bukit Jalil, 50500, KL", "Chickenpox", "21/07/2021", "01.00", Dr. Low, False.

12. Call the following function to pass: "P010", "Jameel", "Daud", "Male", 19, "False", 012334981, "15, Port Dickson, 50310, KL", "Tetanus", "18/04/2020", "18.45", Dr. Sofia, False.

Flowchart

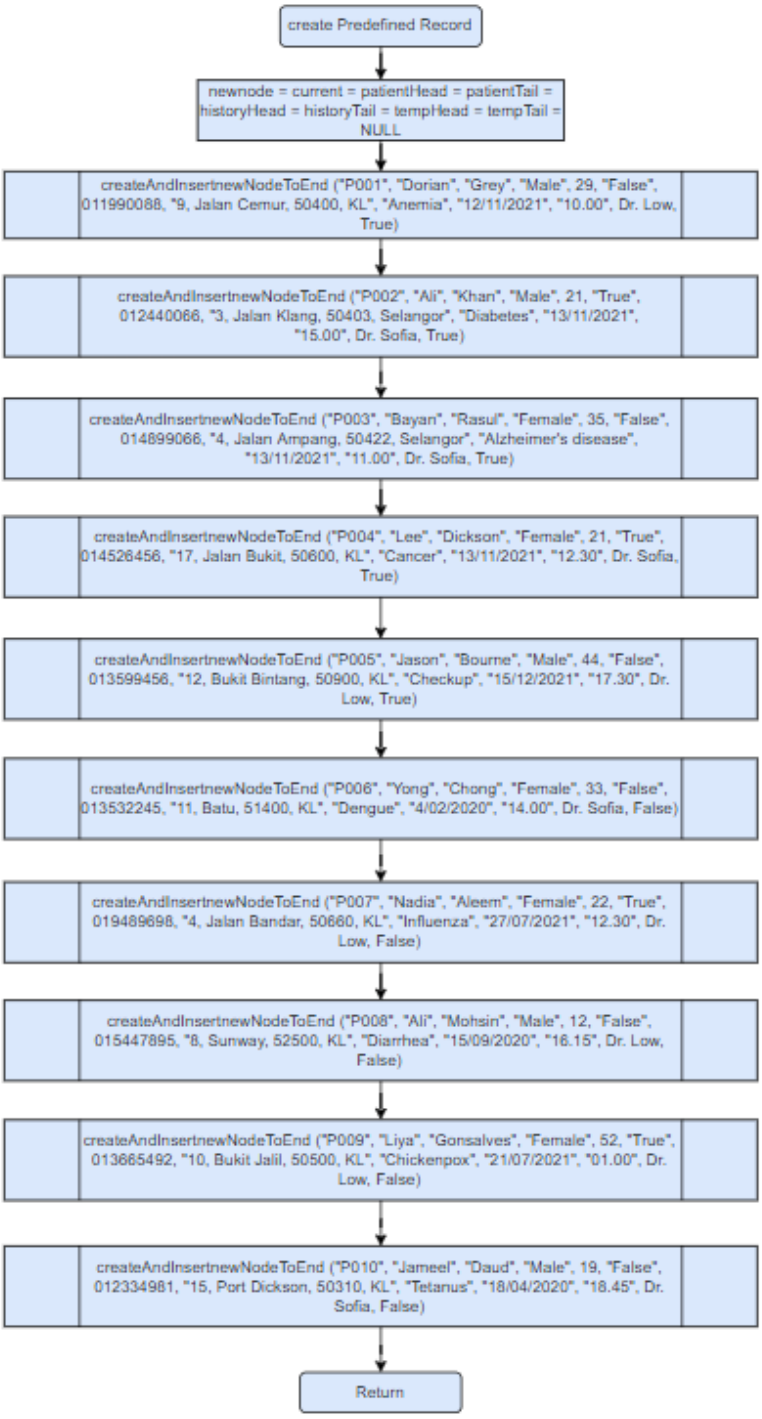

Figure 34; Flowchart of createPredefinedRecords()

Brief Explanation

Above figure demonstrates the process of creating the predefined records for patients. Then it creates and inserts patient's details into the record. Patient record includes patient ID, first name, last name, gender, age, disable, phone number, address, sickness, date of visit, time of visit, doctor name.

nurseLogin()

Algorithm

1. Create a nurse login.
2. Program requests for the password.
3. Read password.
4. Check if the password for the nurse is correct, then calls the nurse() function.
5. Else check if the password is wrong, it prints the message with the options: “incorrect password. 1.try again. 2.exit”.
6. Read user's option
7. If option is 2 then exit the program, else bring user back to enter the password.

Flowchart

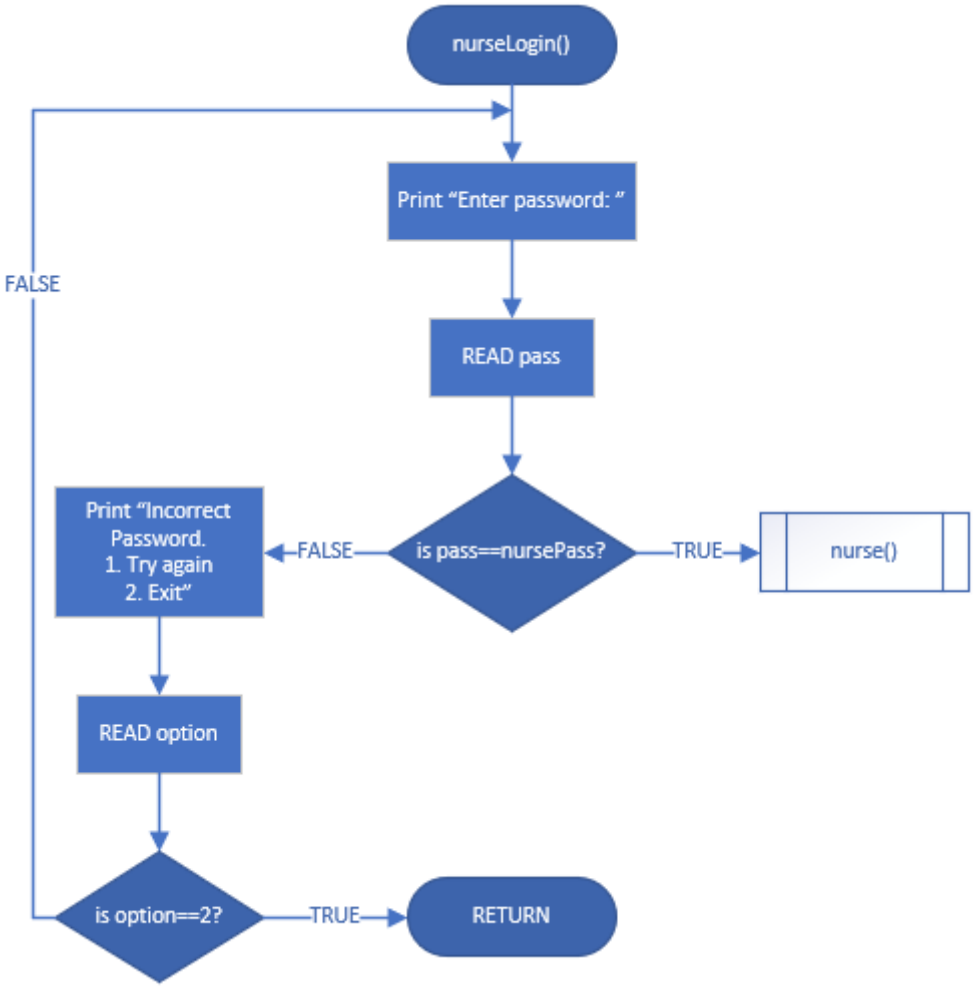

Figure 35; Flowchart nurselogin()

Brief Explanation

This flowchart demonstrates the login process for the nurse. It requests for passwords from user and checks if it is correct or not. If yes it calls the function, else it will ask user to enter password again or exit the program.

doctorLogin()

Algorithm

1. Create a doctor login.
2. Program requests for the password.
3. Read password.
4. Check if the password for the doctor is correct, then calls the doctor() function.
5. Else check if the password is wrong, it prints the message with the options:" incorrect password. 1.try again. 2.exit".
6. Read user's option
7. If option is 2 then exit the program, else bring user back to enter the password.

Flowchart

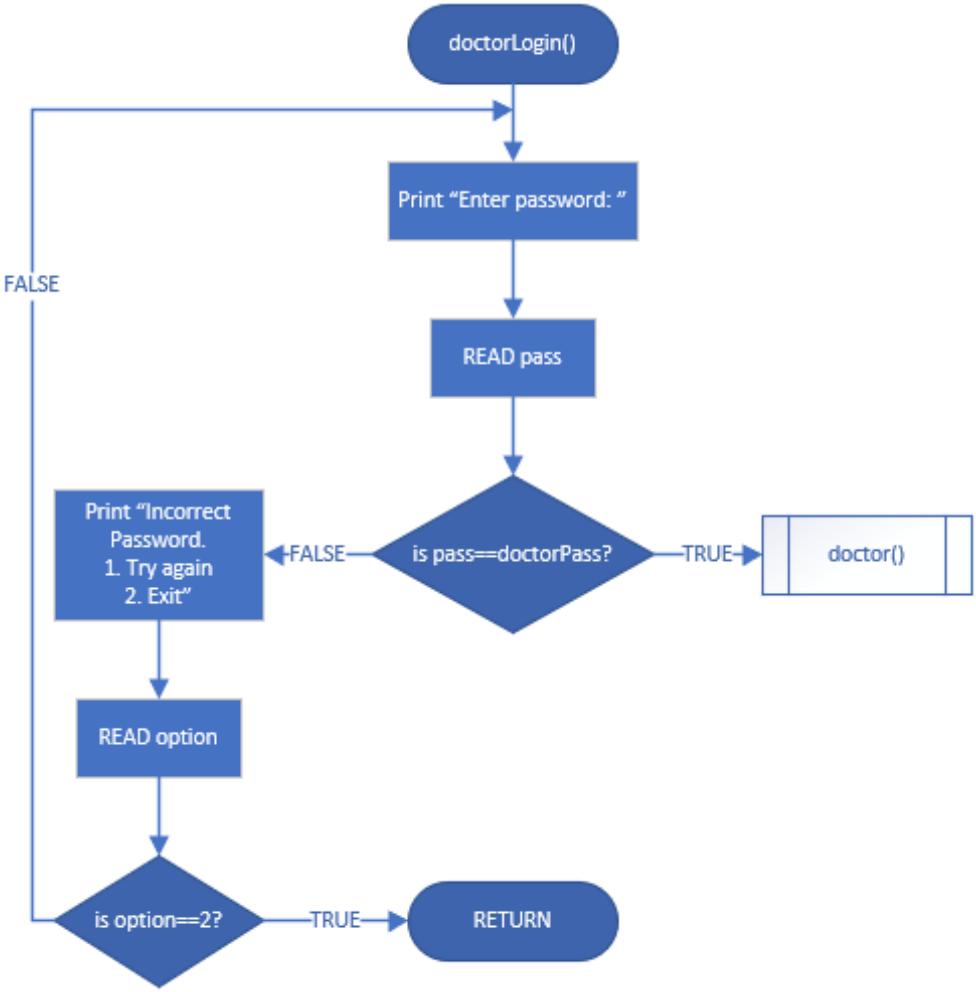

Figure 36; Flowchart doctorlogin()

Brief Explanation

This flowchart demonstrates the login process for the doctor. It requests for passwords from user and checks if it is correct or not. If yes it calls the function, else it will ask user to enter password again or exit the program.

nurse()

Algorithm

1. Start the nurse() function.
2. Display the menu of the functions
3. Use the switch function to make a choice
4. In the first case to register a new patient, program will call addPatientWaitingList() function.
5. In the second case to change patient order, program will call changePatientOrderWaitingList() function.
6. In the third case to view waiting list, program will call viewPatients() function.
7. In the fourth case to call patient to be treated, program will call callPatient() function.
8. In the fifth case to search by patient ID, program will call searchPatientFromWaitingListbyPatientIDorPatientName() function.
9. In the sixth case to sort a waiting list, program will call sortPatientByVusitTimeAscending() function.
10. In the seventh case the program will END
11. If none of the options were chosen, the program will print “Invalid input”.

Flowchart

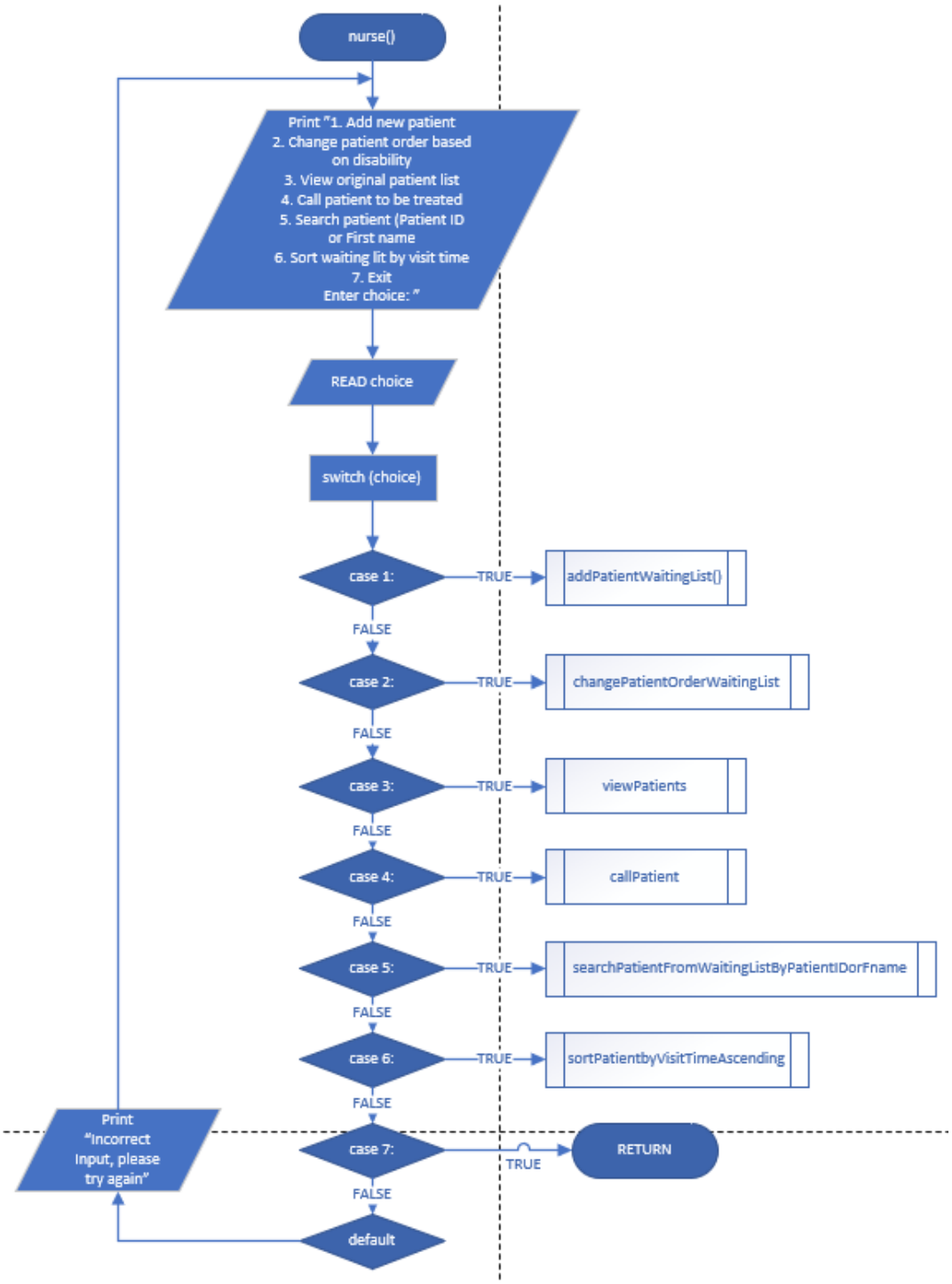

Figure 37; Flowchart of nurse()

Brief Explanation

This flowchart demonstrates the nurse menu and what functions nurse can do under the account. After logging in the nurse will have a different option such as: add new patient, change patient order, view patient list, call patient for treatment, sort waiting list and exit the program. Once the choice is made program will call the function according to the option that has been made.

doctor()

Algorithm

1. Start the program
2. Display the menu of the functions
3. Read the option from menu
4. Use the switch function to make a choice
5. In the first case to view waiting list, program will call viewPatients() function.
6. In the second case to search and modify records, program will call searchPatientandModify() function.
7. In the third case to sort and display records, program will call sortAndDisplayPatientHistoryDescending() function.
8. In the fourth case to search records, program will call searchPatientHistorybySicknessorFname() function.
9. In the fifth case the program will END
10. If none of the options were chosen, the program will print “Invalid input”.

Flowchart

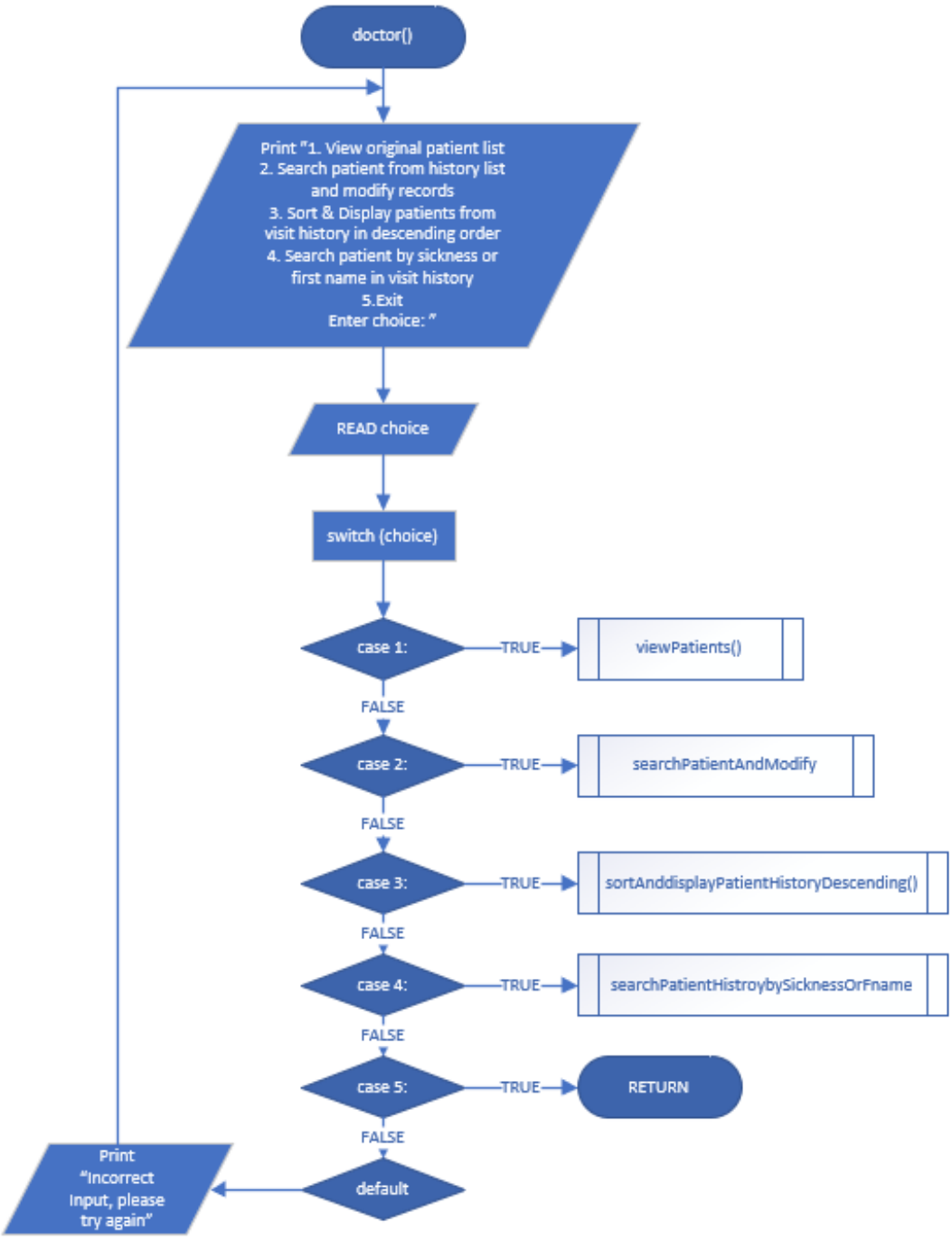

Figure 38; Flowchart of doctor()

Brief Explanation

This flowchart demonstrates the doctor menu and what functions doctor can do under the account. After logging in the doctor will have a different option such as: view original patient list, search patient, and modify the details, sort and display patient data, search patient by sickness and exit. Once the choice is made program will call the function according to the option that has been made

addPatientWaitingList()

Algorithm

1. Start addPatientWaitingList() program
2. Display the message that requires patient ID
3. Accept ID
4. Display the message that requires patient first name
5. Accept fname
6. Display the message that requires patient last name
7. Accept lname
8. Display the message that requires patient gender
9. Accept gender
10. Display the message that requires patient age
11. Accept age
12. Display the message that requires patient first name
13. Accept first name
14. Display the message that requires patient disability, where true is yes and false is no
15. Accept disability
16. Display the message that requires patient phone number
17. Accept pnumber
18. Display the message that requires patient address
19. Accept address
20. Display the message that requires patient sickness
21. Accept sickness
22. Display the message that requires patient visit date
23. Accept vdate
24. Display the message that requires patient visit time
25. Accept vtime
26. Display the message that requires patient doctor name
27. Accept docname
28. Display the message that asks if the patient should be add into waiting list, where true is yes and false is no
29. Accept status
30. Assign the variable “new patient” by combining new data point (node) and patient

31. Assign the variable “id” by adding the patientID into the new data point
32. Assign the variable “fname” by adding the Fname into the new data point
33. Assign the variable “gander” by adding the gender into the new data point
34. Assign the variable “age” by adding the age into the new data point
35. Assign the variable “disability” by adding the disability into the new data point
36. Assign the variable “pnumber” by adding the phoneNumber into the new data point
37. Assign the variable “address” by adding the address into the new data point
38. Assign the variable “sickness” by adding the sickness into the new data point
39. Assign the variable “vdate” by adding the VisitDate into the new data point
40. Assign the variable “vtime” by adding the VisitTime into the new data point
41. Assign the variable “docname” by adding the DoctorName into the new data point
42. Assign the variable “status” by adding the status into the new data point
43. Check if patientHead is equal to NULL, then assign patientHeat to patientTail and newNode then make a return
44. Else take a user to patientTail and return to the previous address, then for the newNode user will go from patientTail to the new address
45. Once patientTail and newNode are equivalent, program can make a return.

Flowchart

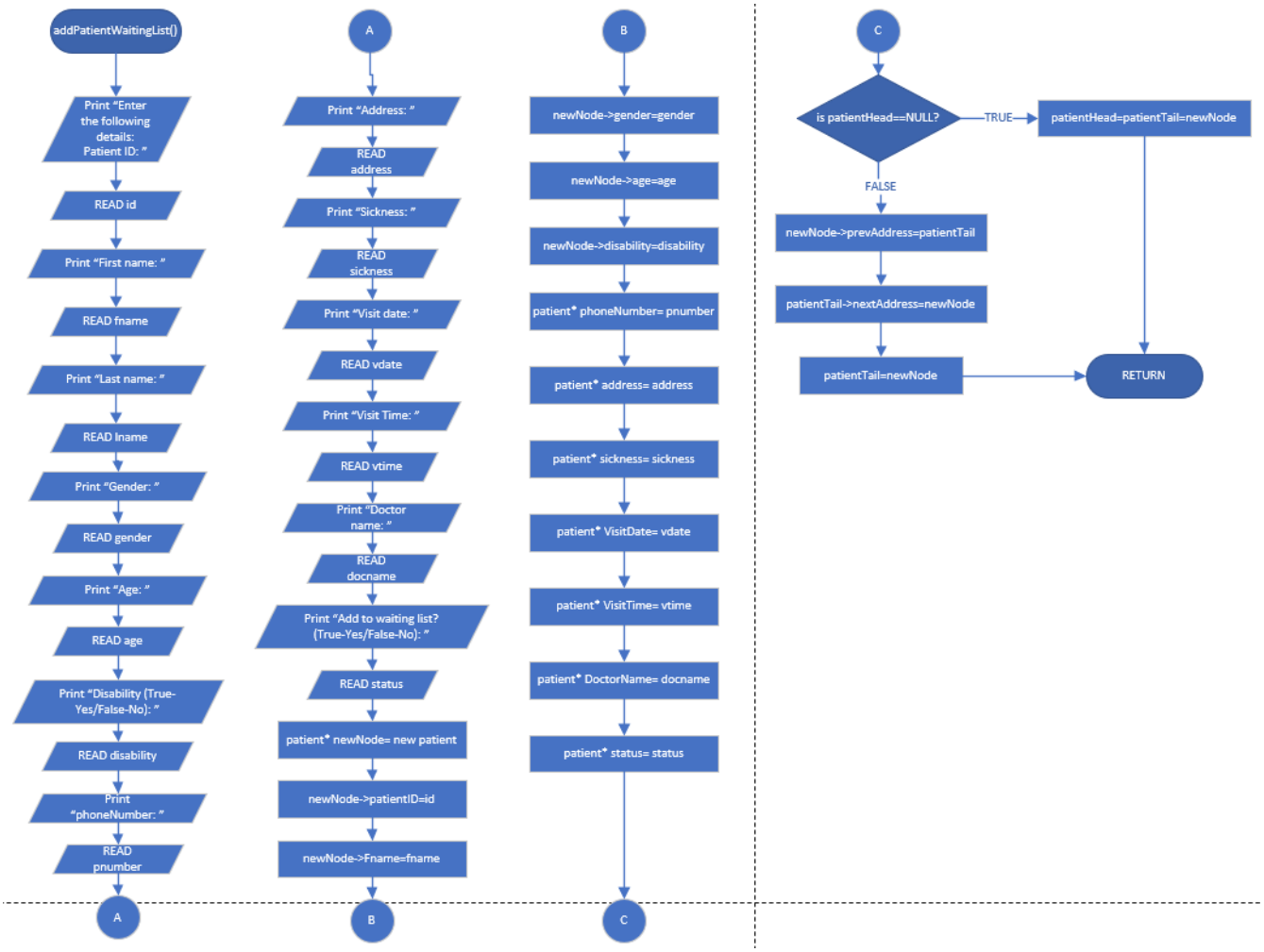

Figure 39; Flowchart for addPatientToWaitingList()

Brief Explanation

Figure above demonstrates how to add a new patient into the waiting list by collecting patient data such as: patient ID, first name, last name, gander, age, disability, phone number, address, sickness, visit date, visit time, doctor name. Once data is collected, it has to be read. After that all the data will be assigned to the new data point. In the end program will move to register new patient.

changePatientOrder()

Algorithm

1. Create a changePatientOrder() function
2. Check if patientHead is equal to NULL, then make a return
3. Else assign current to patientTail
4. Assign temporary to the current, where current is on previous address
5. Check if current patient has the disability, then:
    5.1. Check if next address of value “current” is NULL, then:
        5.1.1. Assign a NULL value to the current value of previous and next address
        5.1.2. Assign a waitingTail value to the current value of previous address
    5.2. Else if the value is not equal to NULL then:
        5.2.1. Assign the value of current (next address) into the current of (previous and next addresses)
        5.2.2. Assign the value of current (previous address) into the current of (previous and next addresses)
    5.3. After the steps 5.1.2 and 5.2.2, assign the patientHead into the current (next address)
    5.4. Assign the current into patientHead (previous address)
    5.5. Assign the NULL into the current (previous address)
    5.6. Assign the current into the patientHead
6. Else of the patient is not disable then assign current value into temporary
7. Check if temporary value is equal to NULL then make a return
8. Else assign temporary value to temporary (previous address) then make a return

Flowchart

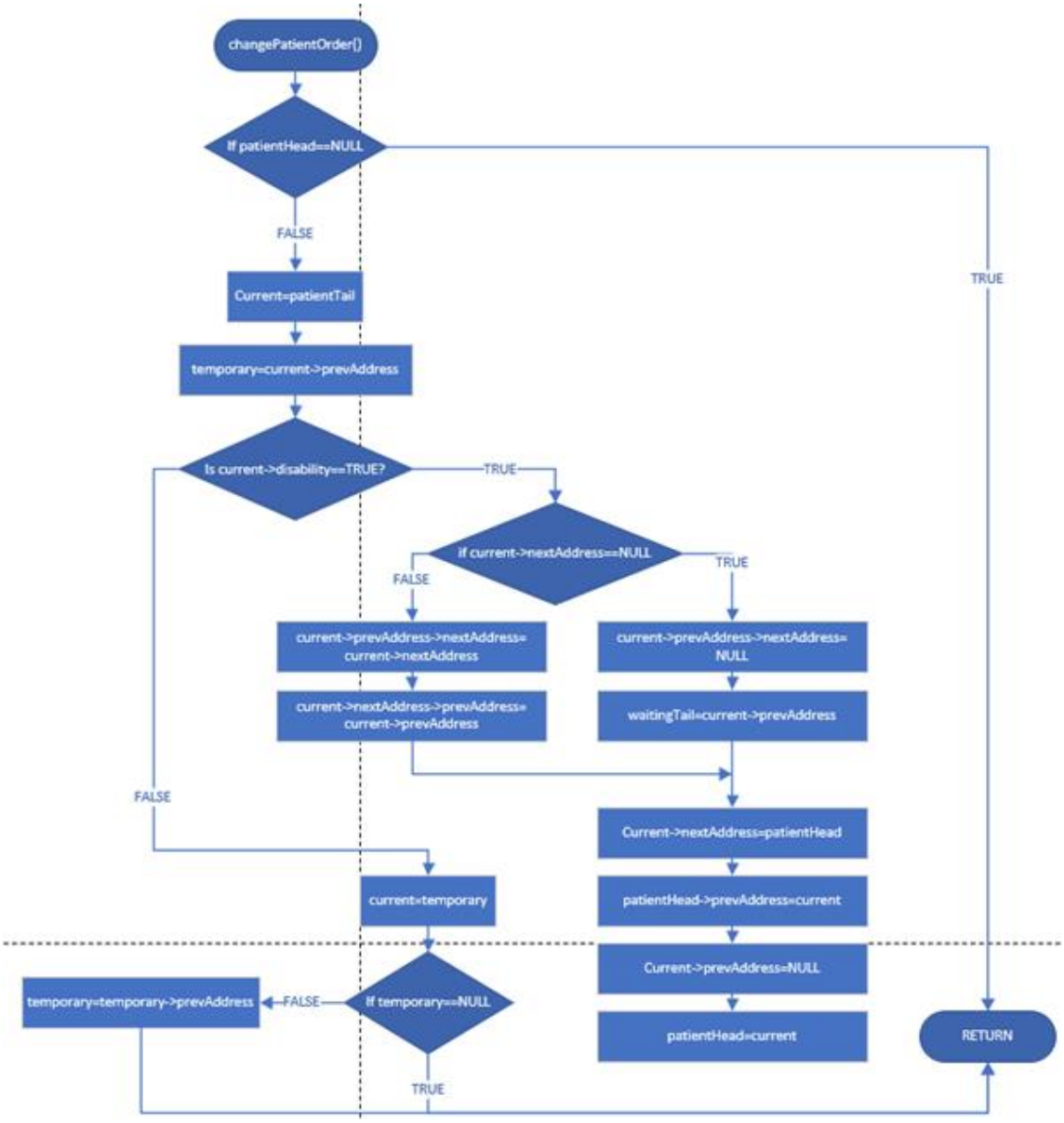

Figure 40; Flowchart for changePatientOrder()

Brief Explanation

This strategy explains how to change the patient order in a waiting list by using if and else statements and current value by changing it into previous or next address. Program will check whether or not patient is disable status and according to that it will proceed to changes. If the status is true, then it will change the order. Else it will keep current request as a temporary.

viewPatient()

Algorithm

1. Start the viewPatients()
2. Assign current to the patientHead
3. Check if current is equal to NULL then call displayTempPatientDetails() function.
4. Assign current where current will go to nextAddress and take user back to check the value of current.
5. Else print “End of list”
6. And return.

Flowchart

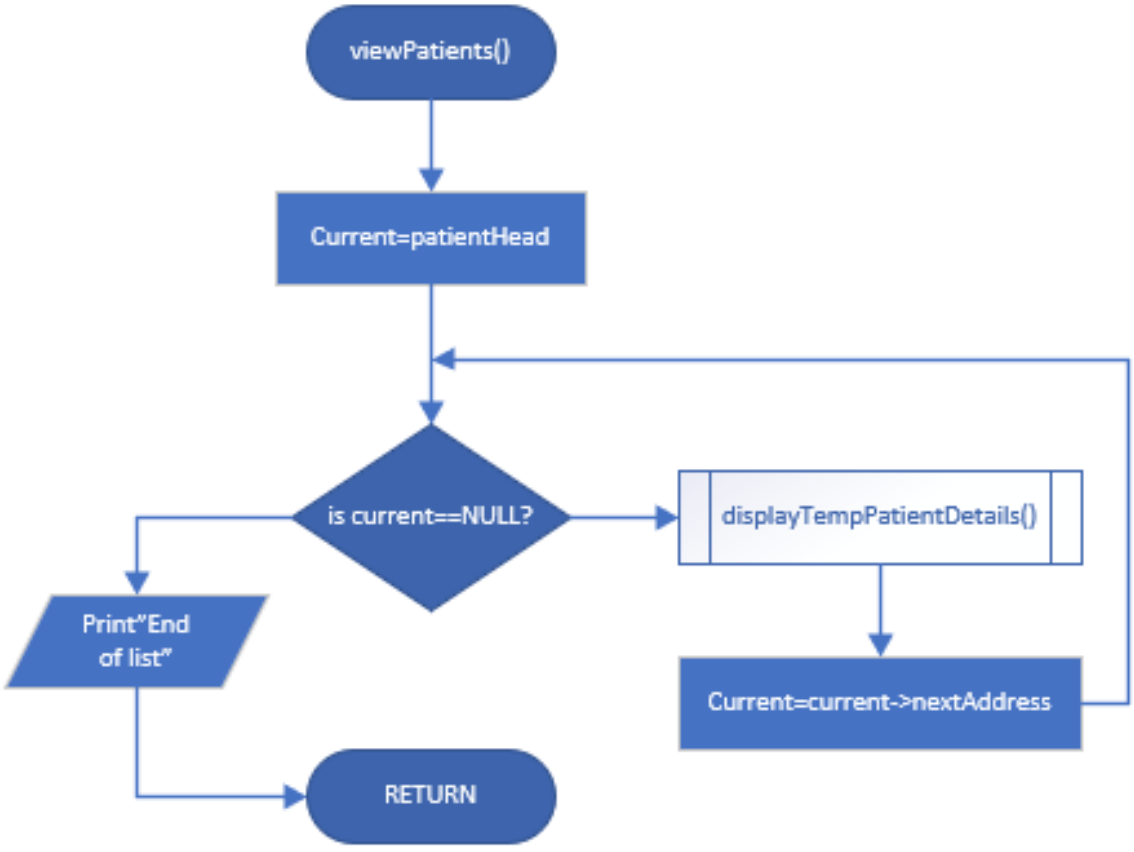

Figure 41; Flowchart for viewPatients()

Brief Explanation

This figure demonstrates how to display the information from the waiting list by calling the function displayTempPatientDetails(). Program assigns the current value to the patientHead and then it checks through the if and else statement weather the current value is equal to NULL. If true it will call the function and changes current value to the next address, else it will print message “End of the list”.

callPatient()

Algorithm

1. Create a callPatient() function
2. Check if patientHead is equal to NULL, then end the program
3. Else assign current value into patientHead
4. Assign the current (previous and next address) value into NULL
5. Assign the patientHead value to patientHead (next address)
6. Check if patientHead is still equal to NULL, then assign patientTail to the NULL
7. Else check historyHead is equal to NULL:
    7.1. If yes, then assign current value to historyTail and historyHead
    7.2. Assign NULL value to current (previous address)
    7.3. Assign NULL value to current (next address)
    7.4. If not, then assign NULL value into current (next address)
    7.5. Assign NULL value into historyTail (next address)
    7.6. Assign historyTail value into current (previous address)
    7.7. Assign current into historyTail
8. After step 7.3 or 7.7 end the program.

Flowchart

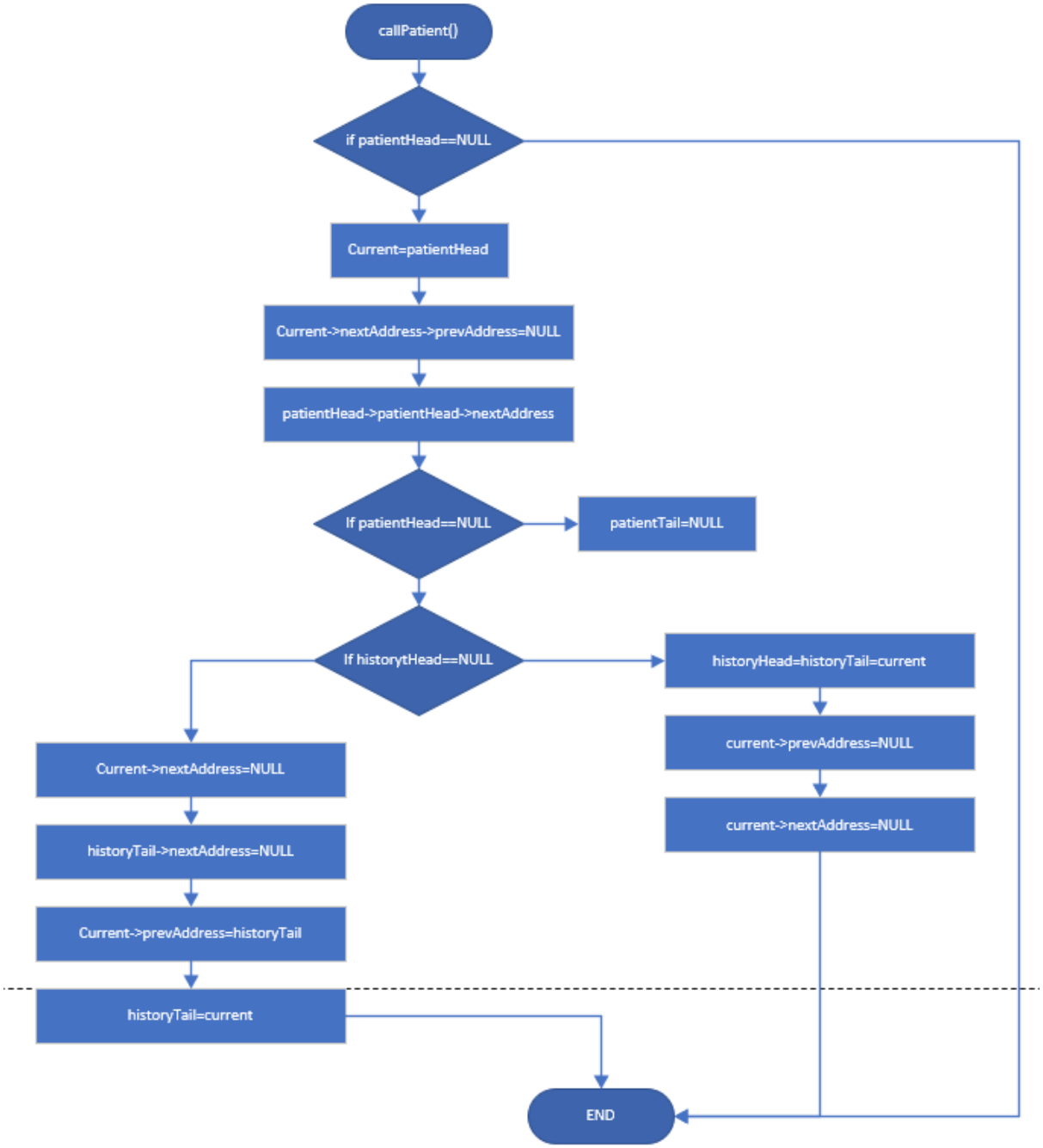

Figure 42; Flowchart for callPatient()

Brief Explanation

This flowchart explains which steps to follow in order to call the patient for treatment according to patient's history records. At first, program checks if the value is not empty, if it is then program will end. Else it will assign the current value to the patientHead and check it through the algorithm above to see if the patient has to be treated or not.

displayPagebyPage()

Algorithm

Step 1: Make the current pointer equal to the tempHead pointer.

Step 2: Read the option entered

Step 3: Check if option is 1, If yes, proceed to step 3, else move to step

Step 4: call the nextpage() function and go back to step 2

Step 5: Clear the screen and display patient details by calling its function and return back to step 2.

Step 6: Check if option is 2, If yes, proceed to step 7, exit the function.

Step 7: call the previouspage() function and go back to step 4.

Flowchart

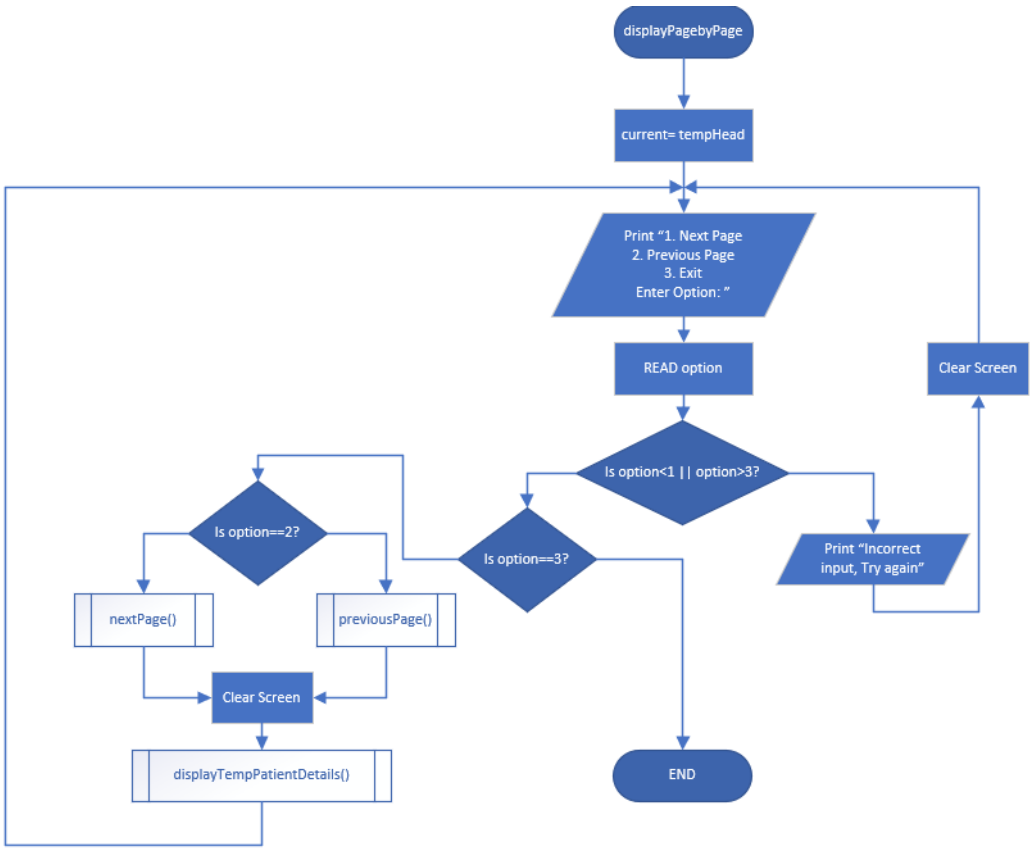

Figure 43; Flowchart for displayPagebyPage

Brief Explanation

Program to display patient details in a page-to-page order. User has 3 inputs to enter which will decide what direction he/she chooses. If option is 1, page will call nextPage() dunction and move forwards, while the opposite happens when option 2 is chosen. They will ultimately call the display temporary patient details to show the data they chose to look at.

createAndInsertnewNodeToEnd(PatientID,Fname, Lname, gender, age, disability, phoneNumber, address, sickness, VisitDate, VisitTime, DoctorName, status)

Algorithm

Step 1: Create a new patient node and its address is assigned to newNode pointer.

Step 2: Fill in the patient details and assign nextAddress andprevAddress to NULL for the first node

Step 3: Check if status is TRUE. If true, continue step 4. Else, go to step 8.

Step 4: Check if the patientHead is NULL. If it is NULL, assign newNode to both patientHead and patientTail and end function. Else, continue to step 5.

Step 5: Allocate the patientTail to prevAddress of the newNode.

Step 6: Allocate the newNode to the nextAddress of patientTail.

Step 7: Allocate the newNode to patientTail and exit the function.

Step 8: Check if status is FALSE. If true, proceed to step 9.

Step 9: Check if historyHead is NULL. If yes, then assign a newNode to both historyHead and historyTail and end the function. Else, proceed to step 10.

Step 10: Assign historyTail to the prevAddress of the newNode.

Step 11: Assign a newNode to the nextAddress of historyTail.

Step 12: Assign a newNode to historyTail and finally end the function.

Flowchart

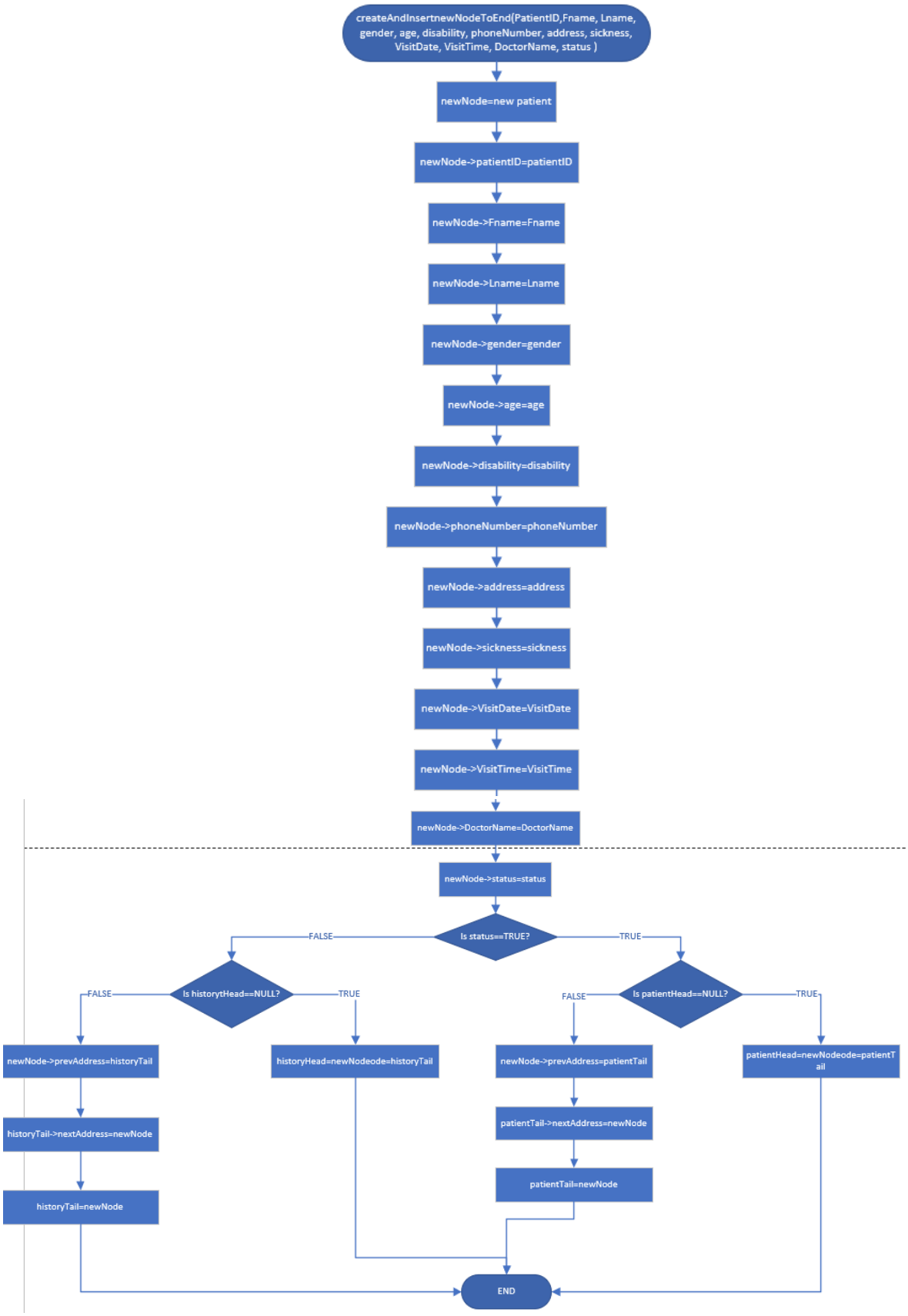

Figure 44; Flowchart for creating new patient

Brief Explanation

The function above aims to simply enter and store the patient details. Status decides if the patient will be in the waiting list or the history list.

nextPage()

Algorithm

Step 1: Check if nextAddress of current is not NULL. If it is not NULL, it will move on to Step 2. If it is NULL, it will move on to Step 3.

Step 2: Make current pointer to point to its next node and exit the function,

Step 3: Print “End of list”.

Step 4: It will exit out of the function.

Flowchart

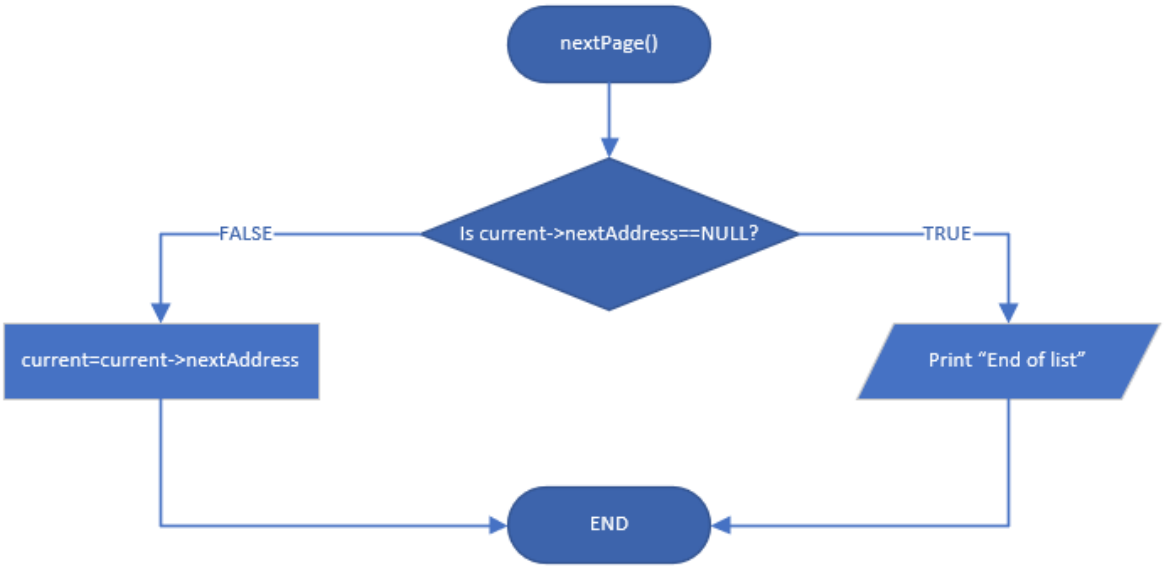

Figure 45; Flowchart for nextPage()

Brief Explanation

The figure above shows the flowchart for the nextPage function. The purpose of this function is to move the current pointer towards the next node. The function is started off by checking whether or not the nextAddress of the current is not NULL. If it is not NULL, then the current pointer will point it to the next node and the function will end. If it is NULL, then the user will receive a message that says “End of list” which indicates that they have reached the end of the list and the function will end.

previousPage()

Algorithm

Step 1: Check if the prevAddress of the current is not NULL, if it is not NULL, then it will move on to Step 2. If it is NULL, it will move on to Step 3.

Step 2: Make the current pointer to point to its previous node and exit the function.

Step 3: Print “End of list”.

Step 4: It will exit the function.

Flowchart

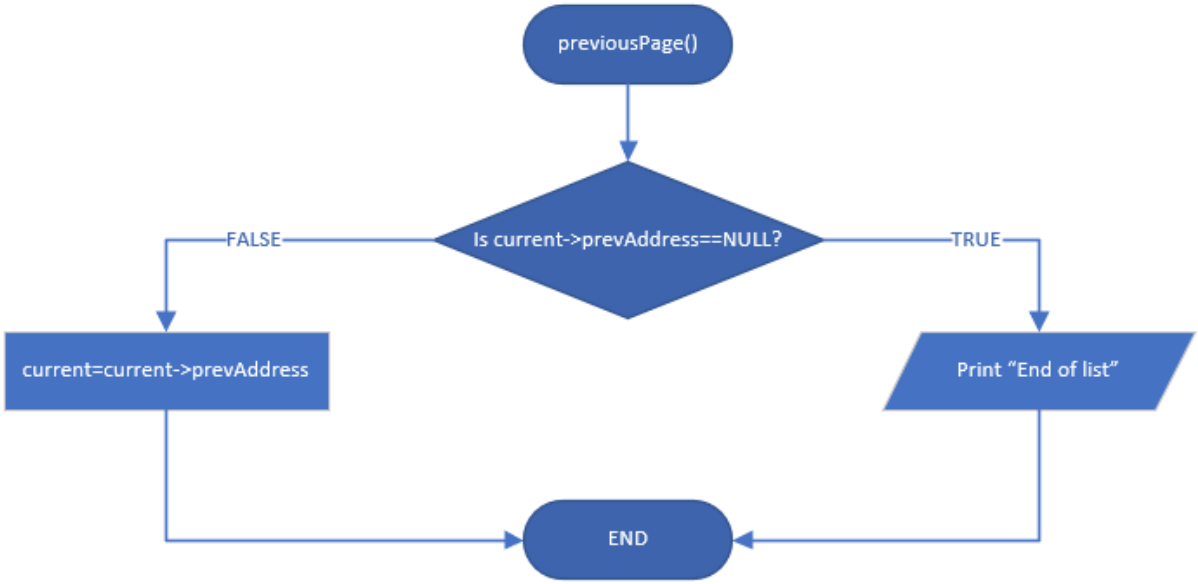

Figure 46; previousPage()

Brief Explanation

The flowchart above is the flowchart for the previousPage function. The function is used to move the current pointer to its previous node. The function is started off by checking if the prevAddress of current is not NULL. If it is not NULL, the current pointer then point to its previous node and the function will come to an end. If the current is NULL, the user will receive a message saying, “End of list” which indicates the system has reached the end of the list and the function will come to an end.

searchPatientID()

Algorithm

1. Creat searchPatientID
2. Call function searchPatientFromWaitingListByPartuent ID()
3. Call function displayPatientDetails()
4. End the program

Flowchart

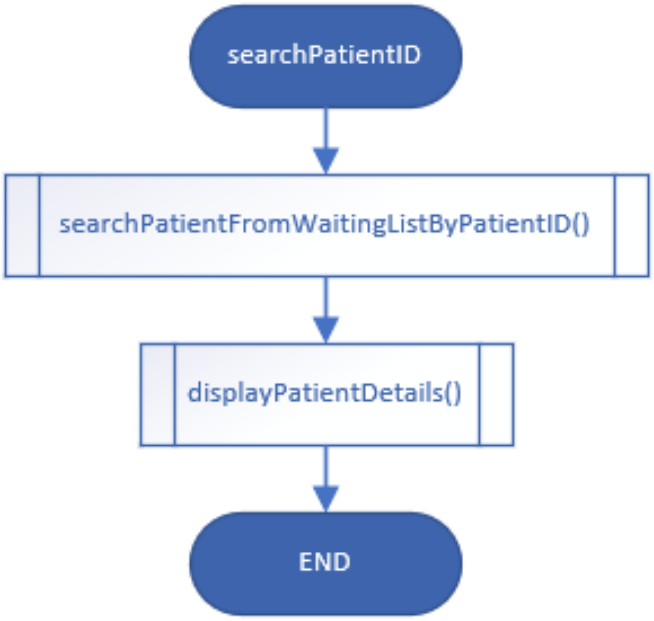

Figure 47; searchPatientID()

Brief Explanation

Figure above demonstrates how to search by patient's ID and see patient's details. There will be two call functions used such as: searchPatientFromWaitingListByPatientID() and displayPatientDetails() in order to search and disaplay the result.

searchPatientHistorybySicknessOrFname()

Algorithm

Step 1: Display a list of search options for the user to choose from.

Step 2: Prompt the user to choose one of the options.

Step 3: Prompt the user to enter in the search keyword.

Step 4: If the user chooses the third option, it will exit out of the function. If the option they select is not 3, then it will proceed to Step 5.

Step 5: if the user chooses to search the patients based on sickess, the generalSearchPatient() function is called and will pass “sickness” , search, historyHead as arguments.

Step 6: If the user chooses to search patients based on first name, the generalSearchPatient() function is called and will pass “Fname”, search, historyHead as arguments.

Step 7: If the user didn’t select any of the options, it will move back to Step 1.

Flowchart

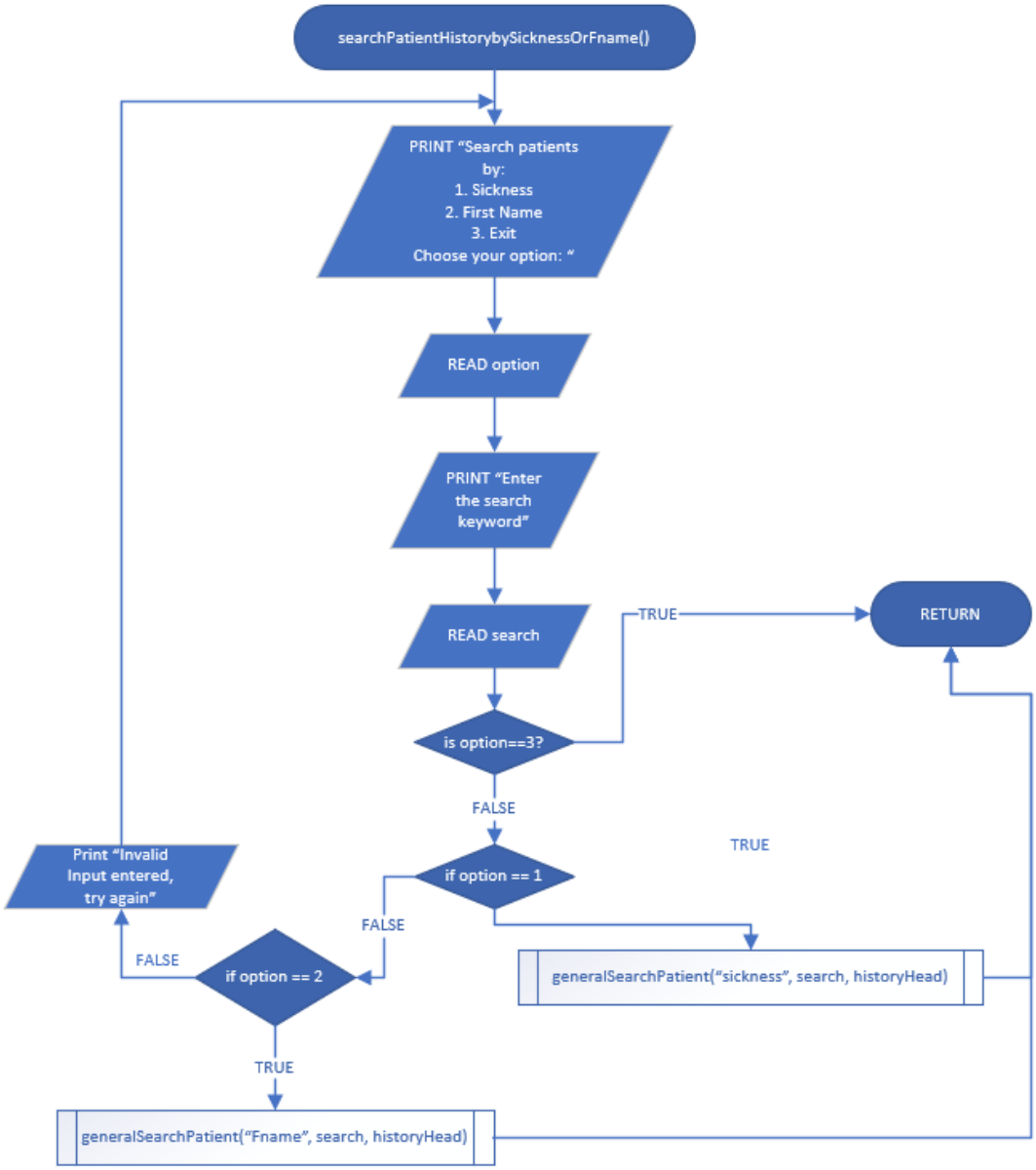

Figure 48; Flowchart for searchPatientHistorybySicknessOrFname()

Brief Explanation

The flowchart above is the flowchart for the searchPatientHistorybySickessOrFname function. This function is used by the doctors to search for the patients based on a keyword for either the sickness or first name. The flowchart begins by displaying a few options which are, sickness, first name and exit. The user is then prompted make a choice from one of the options. Then, the user if prompted to enter the search keyword. The value of choice is then checked to see if it equals to 3. If it equals to 3, the function will come to an end. The value of choice is then checked to see if it equals to 1. If it equals to 1, the generalSearchPatient function will be called by passing “sickness”, search, historyHead as arguments. If the choice equals to 2, the generalSearchPatient function will be called by passing “Fname”, search, historyHead as arguments. If the choice is neither 1 or 2, an error message will be displayed, and the flow will move back to the printing search options. After the generalSearchPatient function has been called, the function will come to an end.

generalSearchPatient(searchType, search, head)

Algorithm

Step 1: Get searchType and search.

Step 2: Get the head pointer which points to the first node of the specified list.

Step 3: Make current pointer to point to the node pointed by the head pointer.

Step 4: In a while loop, it is always checked whether the current is not NULL. If the current is NULL, it will exit out of the loop. If the current is not NULL, it will move on to Step 5.

Step 5: Check if seachBy is equal to “Fname”. If it is, it will move on to Step 6. If it is not, it will move on to Step 7.

Step 6: Check if search is found in the Fname of the current node. If it is, it will move on to Step 9. If it is not, it will move on to Step 10.

Step 7: Check if searchType is equal to “sickness”. If it is, proceed to Step 8. If it is not, proceed to Step 10.

Step 8: Check if search is found in the sickness of the current node. If it is found, it will move on to Step 9. If not, it will move on to Step 10.

Step 9: The details of the current node are displayed by calling the displayPatientDetails function by passing the current pointer as an argument.

Step 10: Make the current pointer to point to the next node of the list and proceed back to Step 4 to check if the current is still not NULL.

Flowchart

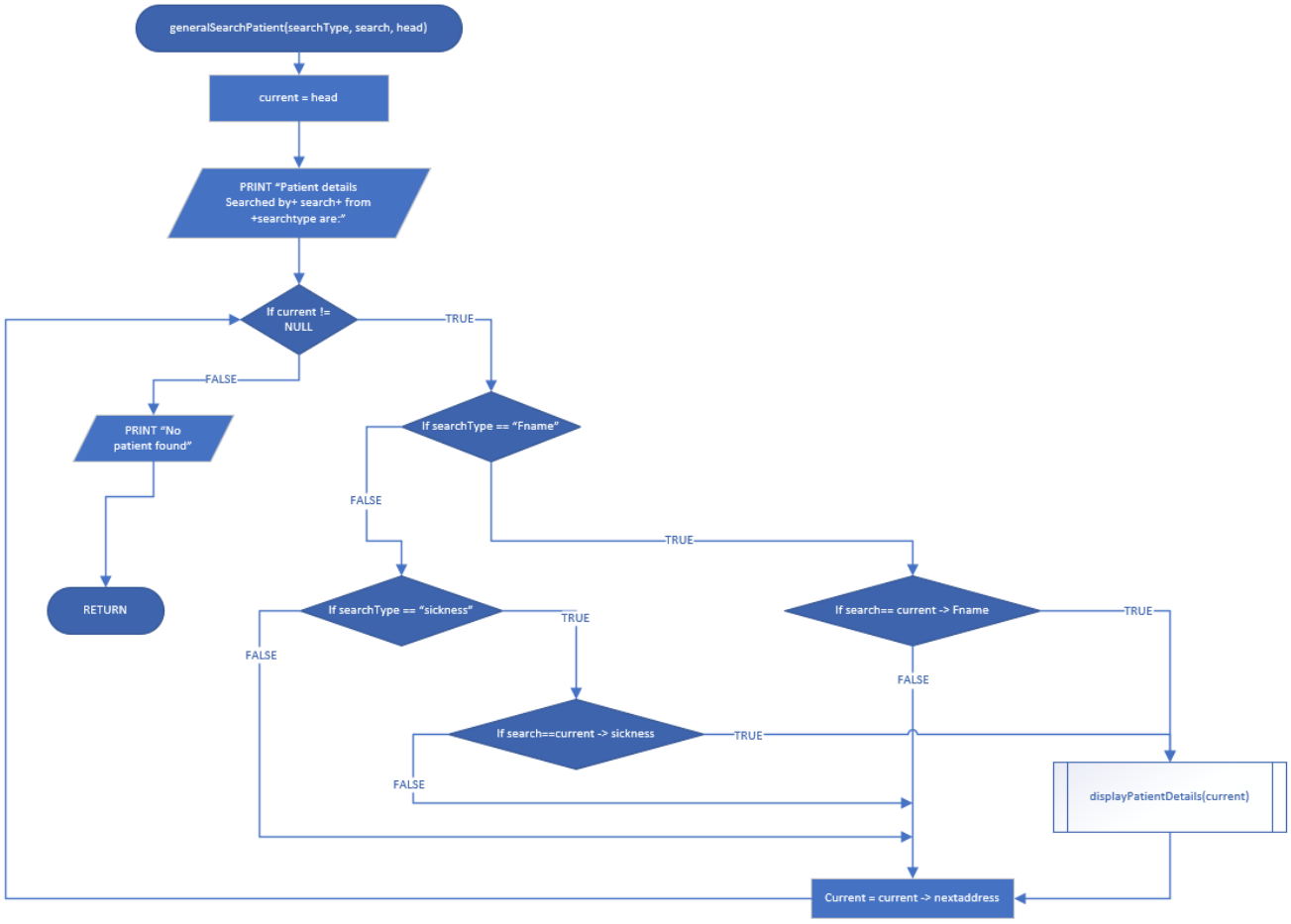

Figure 49; Flowchart for searching all patients

Brief Explanation

The flowchart indicated how the patients will be searched via Sickness or First name. The algorithm used is a linear search. Parameters include the type of search (Fname/sickness), search word inputted and head to go through the details.

searchAndModify()

Algorithm

1. Create searchAndModify() function
2. Display the request message to enter patient ID
3. Accept the id
4. Assign current to head
5. Check if current is equal to NULL, then print “Incorrect id, try again” and send user to main menu
6. Else check if id is equal to current (patient id), if false then assign current to current (next address) and check again if current is equal to NULL
7. If true, call the function displayPatientDeatils()
8. Display the menu with option
9. Read user’s option
10. Use the switch case function
11. In fist case:
    1. Display “Update first name:”
    2. Read new
    3. Assign new to current(fname)
12. In second case:
    1. Display “Update last name:”
    2. Read new
    3. Assign new to current(lname)
13. In third case:
    1. Display “Update phone no:”
    2. Read new
    3. Assign new to current(phoneNumber)
14. In fourth case:
    1. Display “Update address:”
    2. Read new
    3. Assign new to current(address)
15. In fifth case:
    1. Display “Update sickness:”
    2. Read new

3. Assign new to current(sickness)

16. In sixth case:
    1. Display “Thank you”
    2. End program
17. In case the option is not correct, by default:
    1. Display “Incorrect input”
    2. Send user to main menu to make a choice again
18. After the step 11.3, 12.3, 13.3, 14.3, 15.3 call function displayPatientDetails()
19. Display “Continue updating? (Y/N) Choice:”
20. Read the choice
21. Check if choice is equal to “N”, then end the program
22. Else check if choice is equal to “Y” then send user to main menu
23. Else display “Invalid input” and send user to step 19.

Flowchart

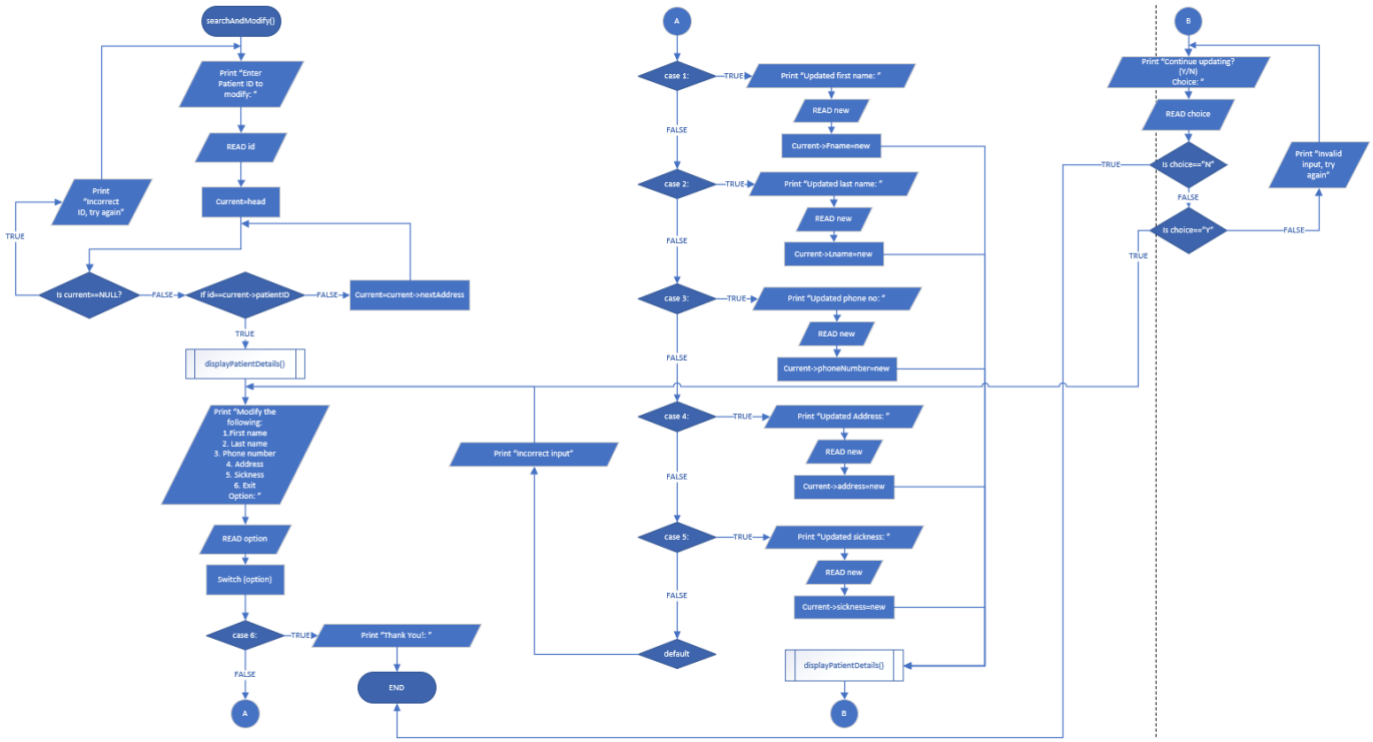

Figure 50; Flowchart for searchAndModify()

Brief Explanation

This figure explains how to search the patient and modify patient's details. Program requests patients ID and after that it has few options of the data that has to be modified such as: first name, last name, phone number, address, sickness. In case the input is wrong, by default program will let the user to try again. After any of the modification is done the program will call the function displayPatientDetails(). In the end program will ask the user if there is any other modifications needed. If not, it will end the program, else it will take user to main menu to make an option again.

addPatientEndOfTempList()

Algorithm

Step 1: Check if the tempHead is NULL. If the tempHead is NULL, newNode is assigned to tempHead and the tempTail and it will exit the function. If the tempHead is not NULL, it will move on to Step 2.

Step 2: TempTail is assigned to newNode -> prevAddress.

Step 3: newNode is assigned to tempTail -> nextAddress.

Step 4: newNode is assigned to tempTail.

Flowchart

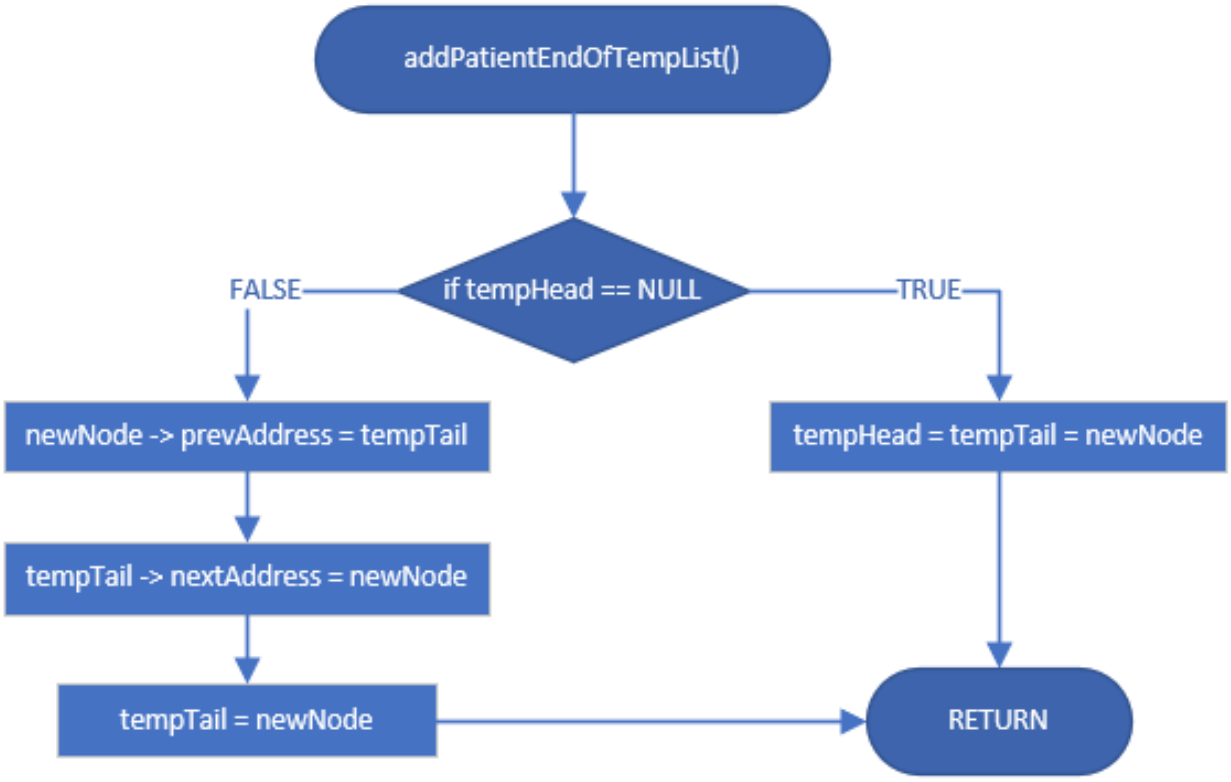

Figure 51; Flowchart for addPatientEndOfTempList()

Brief Explanation

The flowchart above shows the flowchart for the addPatientEndOfTempList function. The use of this function is to insert newly created nodes to the end of the temp list. The function begins by checking whether the tempHead equals to NULL. If the tempHead equals to NULL, the address of the newNode will be assigned to the tempHead and the tempTail pointer of the list. After that, the function will come to an end. But if the tempHead is not NULL, the address of the previous node of the temp list will be

assigned to the prevAddress pointer of the newNode. Then, the address of the newNode will be assigned to the nextAddress of the tempTail. Lastly, the address of the newNode will be assigned to the tempTail pointer and then the function will come to an end.

deleteTempList()

Algorithm

Step 1: Check if the tempHead is NULL. If the tempHead is NULL, initialize the tempTail pointer to NULL and exit the function. If the tempHead is not NULL, move on to Step 2.

Step 2: Make current pointer point to the node pointed by the tempHead pointer.

Step 3: Make tempHead pointer to point to the next node of the temp list.

Step 4: Delete the node that is pointer by the current and procced to move back to Step 1.

Flowchart

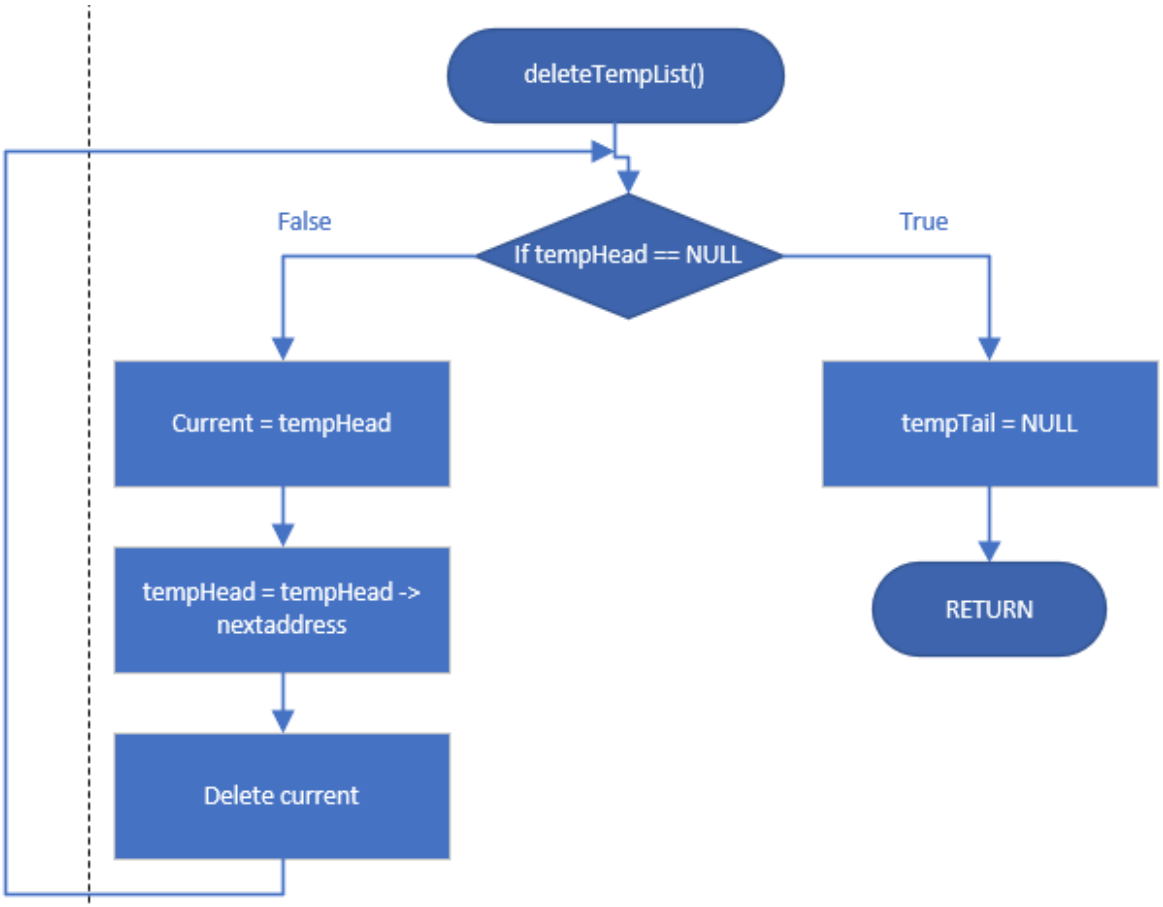

Figure 52; Flowchart for deleteTempList()

Brief Explanation

The flowchart above describes the flowchart for the deleteTempList function. This flowchart is created to delete all of the nodes from the temp list from the memory. The function begins by checking whether the tempHead is NULL. If it is NULL, the tempTail will be initialized to NULL and the function will come to an end. But if the tempHead is not NULL, the current pointer will point towards the node pointed by the tempHead pointer. The tempHead will then be made to point towards its next node and finally, the node that is pointed by the current pointer will be deleted. Once the node is deleted, the function will check again whether or not the tempHead is NULL.

searchPatientFromWaitingListByPatientID()

Algorithm

Step 1: Get the head pointer that points to the first node of the specified list.

Step 2: Make the current pointer point to the node pointer by the head pointer.

Step 3: Check if the id is 0. If it is false, move on to Step 4. If it is true, end the function.

Step 4: Check if current is NULL. If it is not NULL, it will move on to Step 5. If it is NULL, it will end the function.

Step 5: Check if the search is equal to the PatientID of the current node. If it is equal, return the content of the current pointer which is the address of the node and the exit function. If it is not equal, then proceed to Step 6.

Step 6: Make the current pointer point to the next node of the list.

Flowchart

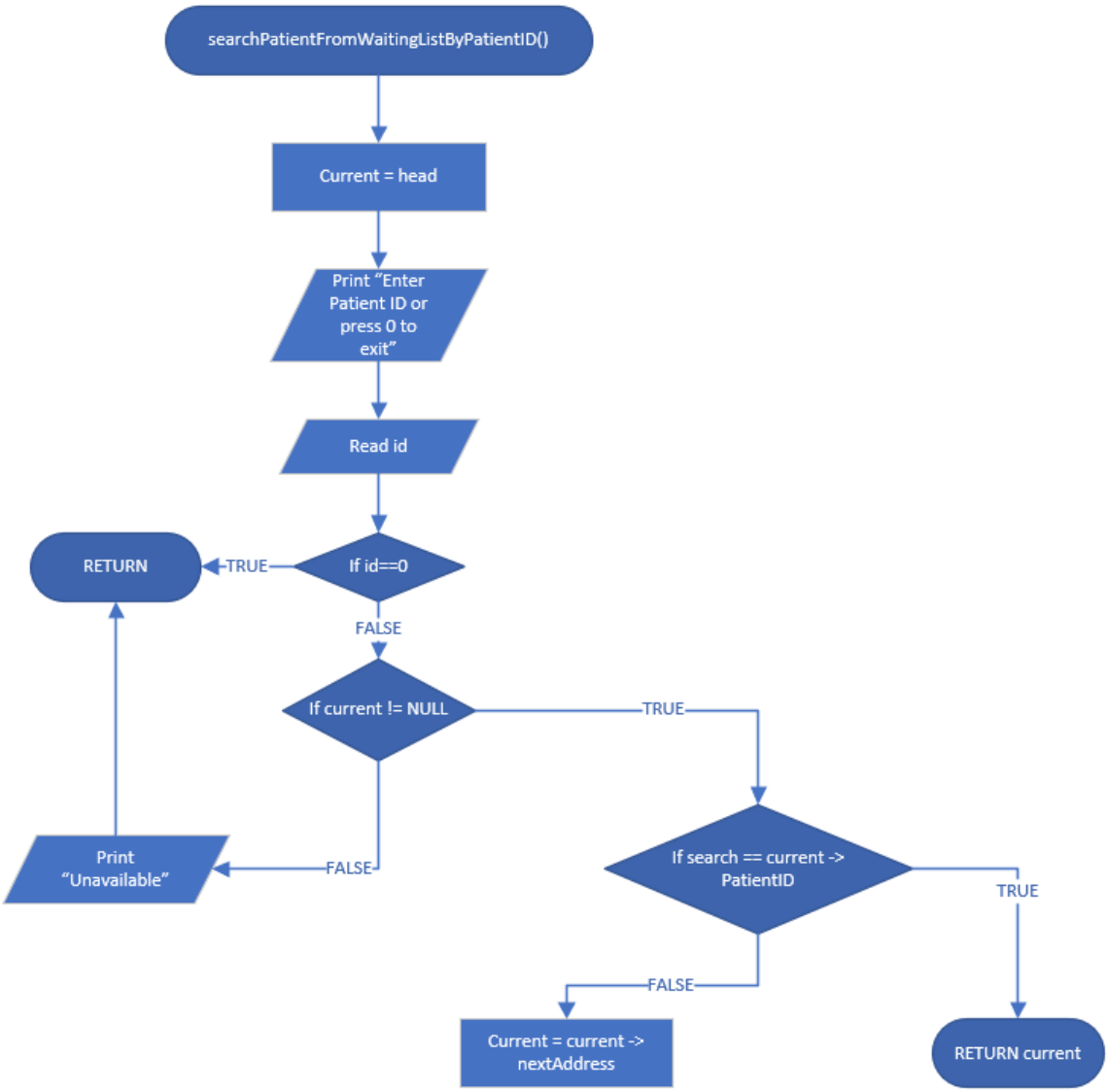

Figure 53; Flowchart for searchPatientFromWaitingListByPatientID()

Brief Explanation

The flowchart above is the flowchart to searchPatientFromWaitingListByPatientID function. This function is used to search patients from the waiting list based on the Patient ID. The user is asked to input the Patient ID into the system. If the ID is equal to 0, the function will come to an end. But if the function is not equal to 0, the system will check if the current is not NULL. If the current is not NULL, the search will be compared to the PatientID of the current node and if there is a match, the content of the current pointer would be returned. If there isn't a match, the current pointer will point to the next node of the list. The flowchart will continue to run until there are no more patients to search from in the list.

displayPatientDetails()

Algorithm

Step 1: Display Patient ID of the current patient.

Step 2: Display First Name of the current patient.

Step 3: Display Last Name of the current patient.

Step 4: Display Gender of the current patient.

Step 5: Display Age of the current patient.

Step 6: Display Disability of the current patient.

Step 7: Display Phone Number of the current patient.

Step 8: Display Address of the current patient.

Step 9: Display Sickness of the current patient.

Step 10: Display Visit Date of the current patient.

Step 11: Display Visit Time of the current patient.

Step 12: Display Doctor Name of the current patient.

Flowchart

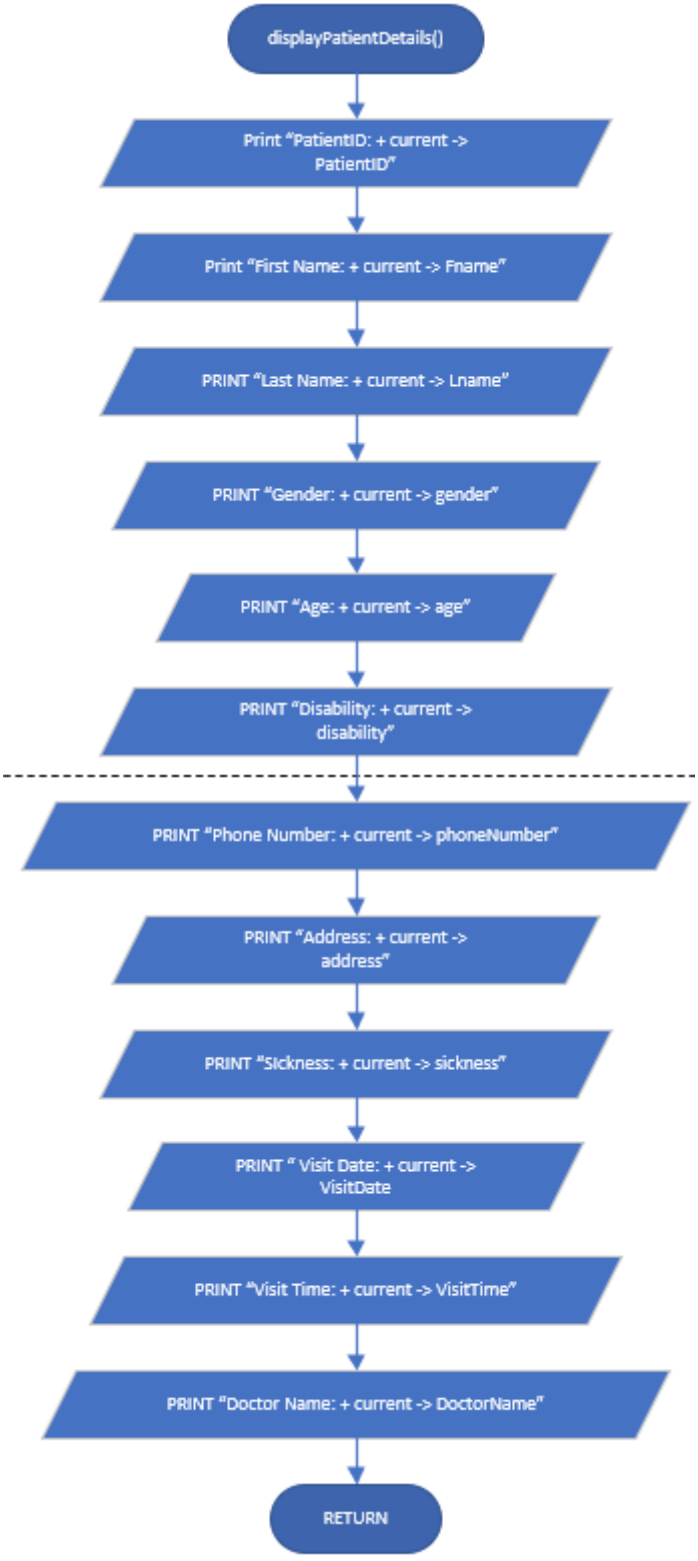

Figure 54; Flowchart for displayPatientDetails()

Brief Explanation

The flowchart above shows the flowchart for the displayPatientDetails function. This function is made to display patient details. The flowchart begins by displaying the PatientID which is then followed by the first name, last name, gender, age, disability, phone number, address, sickness, visit date, visit time and the doctor's name.

createTempPatientDetails()

Algorithm

Step 1: Get head pointer.

Step 2: Make current pointer point towards the node pointed by the head pointer.

Step 3: Check if the current is NULL. If it is NULL, the function will end. If it is not NULL, it will move on to Step 4.

Step 4: Create a newNode and populate the newNode will the details from the current node, excluding the nextAddress and the prevAddress.

Step 5: The addPatientEndOfTempList function is called.

Step 6: The current pointer is made to point to the next node of the list and proceed back to Step 3.

Flowchart

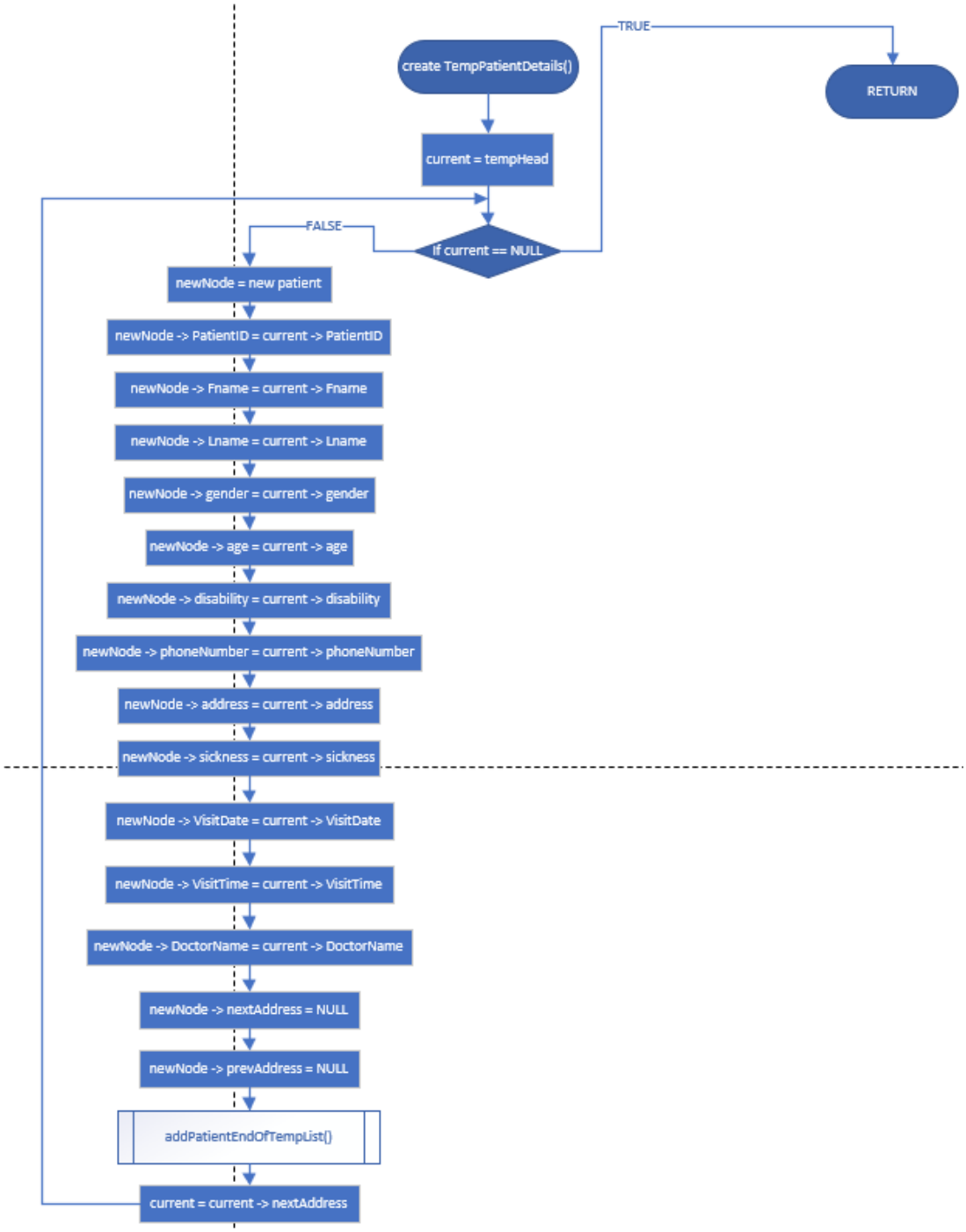

Figure 55; Flowchart for TempPatientDetails()

Brief Explanation

The flowchart above is a flowchart of the create TempPatientDetails function. The reason for this function is to create a temporary list based on pre-existing lists which are the waiting list, or the patient visit history list. A new patient node will be created and the address of it will be stored into the newNode pointer variable. Once that is created, the details of the newNode, excluding the newAddress and the prevAddress will be set to have the same details as the current node. The addPatientEndOfTempList function will be called once each data member of the newNode has been added

Justification of Algorithms

Justification of Searching Algorithms

The searching algorithm that was chosen for this assignment is the linear search algorithm. Linear search is a search that finds an element in the list by searching the element sequentially until the element is found in the list. It is easy to use or in other words, less complex due to the elements in linear search being arranged in any order. As the linear search does not require the list to be sorted, additional elements can be added and deleted. As other searching algorithms may have to reorder the list after insertions or deletions, this may sometimes mean that a linear search will be more efficient (Linear Search vs Binary Search, 2021)

Justification of Sorting Algorithms

For this assignment, the sorting algorithm that we decided to use is the insertion sort. As the name describes, the sorting is done by using successive insertions of the key element selected by the sorting algorithm at its correct place. As the sorting begins the key element chosen is always the second element so that there can be at least one element at the left of the key element to be compared. After comparison the key element is swapped with the element at its left if required and the key element is changed to the element at the immediate right of the previous key element after that the key element is again compared with the elements at it left and the element is swapped with the key element after comparison if required, if the sorting is done in ascending order then the key element should always be greater than the element at its left if not, then the swapping is performed with key element and vice versa for the descending order (Insertion sort Algorithm, flowchart and C, C++ Code, 2021).

Insertion sort is considered to be one of the easiest and more efficient sorting algorithms compared to other sorting algorithms. We chose the insertion sort as it is very easy to implement and if the elements are already sorted, it does not spend a lot of time in useless operations and instead will deliver a run time of 0(n) (Pathak, 2021).

References

Big-O Algorithm Complexity Cheat Sheet (Know Thy Complexities!) @ericdrowell. (2021). Bigocheatsheet.com. Retrieved September 29 2021, from https://www.bigocheatsheet.com/

Data Structure and Algorithms - Linked List. (2021). Tutorialspoint.com. Retrieved September 22 2021, from https://www.tutorialspoint.com/data_structures_algorithms/linked_list_algorithms.htm

Difference between Singly linked list and Doubly linked list in Java. (2019). Tutorialspoint.com. Retrieved September 25 2021, from https://www.tutorialspoint.com/difference-between-singly-linked-list-and-doubly-linked-list-in-java

Linked List Data Structure. (2021). Programiz.com. Retrieved September 23, 2021, from https://www.programiz.com/dsa/linked-list

Types of linked list - singly linked, doubly linked and circular. Programiz. (n.d.). Retrieved October 15, 2021, from https://www.programiz.com/dsa/linked-list-types.

javapoint. (n.d.). Linear search vs binary search - javatpoint. www.javatpoint.com. Retrieved October 15, 2021, from https://www.javatpoint.com/ds-linear-search-vs-binary-search.

Insertion sort Algorithm, flowchart and C, C++ Code. (2018). Retrieved September 23, 2021, from Includehelp.com website: https://www.includehelp.com/algorithms/insertion-sort-algorithm-flowchart-and-c-cpp-code.aspx

Chankey Pathak. (2021). Insertion Sort :: TutsWiki Beta. Retrieved September 20, 2021, from Tutswiki.com website: https://tutswiki.com/data-structures-algorithms/insertion-sort/