

**A . P . U**  
ASIA PACIFIC UNIVERSITY  
OF TECHNOLOGY & INNOVATION

**GROUP ASSIGNMENT**

**TECHNOLOGY PARK MALAYSIA**

**CT077-3-2-DSTR**

**DATA STRUCTURES**

**UC2102CS(DA)**

**ASSIGNMENT TITLE : DSTR Assignment Report**

**HAND OUT DATE : 08 SEPTEMBER 2021**

**HAND IN DATE : 10 NOVEMBER 2021**

**WEIGHTAGE : 50%**

---

**GROUP MEMBERS : ONG CHENG KEI TP055620**

**KONG KEI ZHONG TP055016**

**WONG JACK YIH TP055436**

**LECTURER NAME : CHONG MIEN MAY**

## Table of Contents

<b>Introduction</b> .....	3
<b>Code Explanation and Justification</b> .....	4
LinkedList Class .....	4
PatientQueue Class .....	12
Node Class .....	19
Doctor Class.....	22
Nurse Class .....	33
Patient Class.....	47
Utility Class .....	54
Main Class .....	64
<b>Screenshot of Inputs &amp; Outputs</b> .....	67
Nurse Users.....	67
Doctor User.....	89
<b>Conclusion</b> .....	129

## Introduction

This report details the structure and process workflow of the entire system, in which it aims to assist Klinik Sulaiman in their daily business processes. The report mainly details the main classes and attributes, their primary purpose and functions, as well as some of the decisions and design implementations being made for the system. From a top-level view of the system, it relies on linear search methods to search specific data within the nodes of the linked list, while implementing a merge sort algorithmic approach to sort the linked list before allowing the user to view the linked list. This program consists of 2 linked lists, the first linked list named patient waiting list and the second linked list named history list, the patient waiting list would require the nurse to register a new patient and enter the patient details into the waiting list, this process would be done in a first in first out approach, however, if the patient is disabled, then they would be added into the queue before all the non-disabled patients. Once the patient has been registered, they will need to wait until the nurse uses its call function to invite the first patient within the patient waiting queue to visit the doctor. The waiting queue would remove the patient from the queue and be add the patient to the history list. In the history list, which is the second linked list within this system, basic patients record data such as names and visit time, etc would already be entered by the nurse, however, the nurse would not have the privilege of entering patient attributes such as medicine information as this is reserved for the doctor to perform diagnosis of the patient. Once the doctor has finished diagnosing the patient, the doctor would enter all the relevant data that is specified previously. In the event that the doctor made a mistake or a patient visits the doctor again in the future, then the system would allow the doctor to have multiple search options to view all existing patient records and allow for record modification. This feature would only be reserved for the doctor and not for the nurse.

## Code Explanation and Justification

In this section, all the classes and code that are used to build the Klink Sulaiman management system will be discussed. In total there are 8 classes and all of the classes declarations are stored in separate header files.

### LinkedList Class

```
class Node;
class Patient;

class LinkedList {
protected :
    Node* head;
    Node* tail;
    int size;

public :
    LinkedList();

    void insertAtFront(Patient* patient);

    void append(Patient* patient);

    LinkedList* search(std::string searchReference, int searchMode);

    void displayList();

    void displayInQueue();

    void reverseDisplayInQueue();

    Node* getHeadReference();
    Node* getTailReference();

    int getSize();
    static LinkedList* concatLists(LinkedList* a, LinkedList* b);
};
```

*Figure 1: System Overview for the Linked List Class*

The primary use for the linked list class is to host and contain all the necessary tools that would be applicable on a linked list. This class is utilised across both the patient queue as well as the patient history list. The linked list has a constructor which initialized the linked list head and tail nodes. The linked list class would also have a size integer to keep track of the number of nodes within the linked list. This class contains 10 functions in total, the functions would be used for both classes, however, some exclusive requirements that are required for the patient

queue list would have an inheritance relationship to the linked list class, which would have its exclusive functions located there.

```
void LinkedList::insertAtFront(Patient* patient) {  
    Node* newNode = new Node(patient);  
    if (head == NULL) {  
        tail = newNode;  
    }  
    else {  
        newNode->setNextNode(head);  
        head->setPreviousNode(newNode);  
    }  
    head = newNode;  
    size++;  
}
```

*Figure 2 Insert at front method*

Figure 2 shows the insert at front method available under the LinkedList class. The insert at the front method takes in a pointer that is pointing to a patient object and proceeds to encapsulate the patient object in a node before adding it into the linked list. The encapsulation process is shown by the code in the first line of the method. Then, if the linked list is empty (head equals NULL), then the node will be set as the head and tail of the linked list. On the other hand, if the linked list contains other nodes, then the newNode will be added to the front of the linked list by using setNextNode and setPreviousNode methods to adjust the links between the nodes. Finally, the size attribute is incremented by one.

```
void LinkedList::append(Patient* patient)
{
    Node* newNode = new Node(patient);

    if (tail == NULL)
    {
        head = newNode;
        tail = newNode;
        size++;
        return;
    }

    tail->setNextNode(newNode);
    newNode->setPreviousNode(tail);
    tail = newNode;
    size++;
}
```

*Figure 3 Append method*

Figure 3 shows the append method under the linked list class. The append method is focused on adding patients at the end of the linked list. Similar to the insert at front method shown in Figure 2, the append method takes in a pointer that is pointing towards a patient object. Since the linked list class does not handle patient objects directly, therefore, the patient objects will be encapsulated inside a node object. Moving on, the append method will check whether the linked list is empty, if it is empty, then no links between the nodes are required to be adjusted when the newNode is added in. Otherwise, the links between the tail and newNode will be changed accordingly to make the newNode the new tail of the linked list. In addition, the method also increments the size of the linked list by one for every newNode.

```
LinkedList* LinkedList::search(std::string searchReference, int searchMode)
{
    LinkedList* results = new LinkedList;
    Node* currentNode = head;
    std::string searchComparator;
    if (head != NULL)
    {
        while (currentNode != NULL)
        {
            switch (searchMode)
            {
                case 1:
                    searchComparator = currentNode->getPatient()->getPatientID();
                    if (searchComparator == searchReference)
                    {
                        results->append(currentNode->getPatient());
                        return results;
                    }
                    break;

                case 2:
                    searchComparator = currentNode->getPatient()->getFirstName();
                    break;

                case 3:
                    searchComparator = currentNode->getPatient()->getSicknessDescription();
                    break;

                case 4:
                    searchComparator = currentNode->getPatient()->getMedicineInformation();
                    break;

                case 5:
                    searchComparator = currentNode->getPatient()->getDoctorName();
                    break;

                default:
                    break;
            }

            if (searchComparator == searchReference)
            {
                results->append(currentNode->getPatient());
            }
            currentNode = currentNode->getNextNode();
        }
    }

    return results;
}
```

*Figure 4 Search method*

The figure above shows the method used to search for patient profiles in the linked list. The method accepts a searchReference parameter to identify what the user is searching for and a searchMode parameter to determine the patient attribute to be searched with. The method creates a new LinkedList object to store the nodes containing the pointer for the patient object that the user is searching for. Then, a Node pointer is created to point towards the head of the current LinkedList object and will be used to traverse the lists later on. The method then checks



if the head of the linked list is empty, where the function will immediately end if it is. Otherwise, a loop will be used and will run as long as the currentNode pointer is not empty. Within the loop, a switch is used to determine which attribute of the patient is read using the searchMode attribute passed in. For example, searchMode 1 is for patient ID, 2 for patient first name, and so on. The attribute of the patient searched is then read and stored in the string variable named searchComparator, which is then compared with the searchReference parameter. If both the variable details matched, the patient object will then be added into the linked list created, and the currentNode will move to the next node and the loop repeats. After reaching the end of the list, the method will then return the linked list created that contains details of all the patient searched. The one exception where the method ends immediately is when the user search for patient ID with searchMode 1, and the patient with matching ID is found. This is because the ID attribute is unique, and there should not be multiple records of patient with the same ID, hence ending the method once the entered ID is found.

```
void LinkedList::displayList()
{
    Node* currentNode = head;
    while (currentNode != NULL)
    {
        std::cout << currentNode->getPatient()->toString();
        std::cout << "-----\n";
        currentNode = currentNode->getNextNode();
    }
}
```

*Figure 5 Display list method*

Figure 5 shows the display list method under the linked list class. The display list method is simple and self-explanatory. The method did not require any parameters and the method will print out every patient's detail by using the toString method in the patient class.

```
Node* LinkedList::getHeadReference()
{
    return head;
}
```

*Figure 6: Get head reference function (Linked List)*

The getHeadReference function is used across the entire program, which primarily serve the function of returning the address of the head node, which allows the program to start traversing the linked list.

```
int LinkedList::getSize() {  
    return size;  
}
```

Figure 7: Get size function (Linked List)

The get size function is used to retrieve the number of nodes within the linked list, this function is primarily used during the registration process of a new patient by the nurse to assign the patient ID.

```
LinkedList* LinkedList::concatLists(LinkedList* a, LinkedList* b) {  
    LinkedList* results = new LinkedList;  
    Node* nodeFromA = a->getHeadReference();  
    Node* nodeFromB = b->getHeadReference();  
    while (nodeFromA != NULL) {  
        results->append(nodeFromA->getPatient());  
        nodeFromA = nodeFromA->getNextNode();  
    }  
    while (nodeFromB != NULL) {  
        results->append(nodeFromB->getPatient());  
        nodeFromB = nodeFromB->getNextNode();  
    }  
    return results;  
}
```

Figure 8 Concat list method

Figure 8 shows the method that is used to concatenate both linked lists together. The concat list method takes in two parameters which are pointers pointing towards two different linked lists. The concat lists method will then return a new linked list that contains all the nodes from two of the linked lists passed into the method.

## PatientQueue Class

```
class Node;
class Patient;

class PatientQueue : public LinkedList {
    Node* lastDisabledPatient = NULL;
public:
    void insertPatient(Patient* patient);

    Patient* getNextPatient();

    int removePatient(std::string patientID);
};
```

*Figure 9: System Overview for Patient Queue*

The primary use of the Patient Queue class is to manage the order and number of waiting patients in the waiting queue. This class is the child class of the Linked list class and inherits all the properties of a linked list. The reason for it being a child class is due to its requirement to add new patient and call the patients to visit the doctor, it is due to these unique requirements in which the linked list class does not provide, hence a separate inheritance is required. This class has 3 functions and will have an attribute to keep track on the address node of the last patient. This class would be only used by the nurse.

```
void PatientQueue::insertPatient(Patient* patient) {  
    if (patient == NULL) {  
        return;  
    }  
    if (patient->isDisabled()) {  
        if (lastDisabledPatient == NULL) {  
            insertAtFront(patient);  
            lastDisabledPatient = head;  
        }  
        else {  
            Node* newNode = new Node(patient);  
            newNode->setPreviousNode(lastDisabledPatient);  
            newNode->setNextNode(lastDisabledPatient->getNextNode());  
            if (lastDisabledPatient->getNextNode() != NULL) {  
                lastDisabledPatient->getNextNode()->setPreviousNode(newNode);  
            }  
            lastDisabledPatient->setNextNode(newNode);  
            lastDisabledPatient = newNode;  
            size++;  
        }  
    }  
    else {  
        append(patient);  
    }  
}
```

*Figure 10 Insert patient method*

Figure 10 shows the insert patient method under the patient queue class. Firstly, this method accepts a pointer that points toward a patient object as a parameter. Then, this method attempts to insert the patient object into the patient queue according to the patient's disabled status. The patient's disabled status can be obtained through the isDisabled method used in the insert patient method as shown in Figure 10. If the patient is disabled, then the patient will be placed before every non-disabled patient in the patient queue. Otherwise, the patient will be placed at the end of the queue. Since the patientQueue class is the child class of the LinkedList class, therefore the append method is inherited and can be used to insert patients at the end of the linked list. On the other hand, the lastDisabledPatient pointer is used to keep track of which patient is the last disabled patient so that a new disabled patient can be added right behind the last disabled patient in the patient queue. This assumes that the lastDisabledPatient pointer is not null, if the lastDisabledPatient pointer is null, this means that there was no prior disabled patient, hence the new disabled patient will be inserted at the front of the patient queue via the insert at front method inherited from LinkedList class. Another thing is that the

lastDisabledPatient will always shift one place to the right when a new disabled patient is joining the queue, in other words, it will point to the new disabled patient that joined the queue.

By using this algorithm, the insertion process for patients regardless of their disabled status will remain in  $O(1)$  as looping through the linked list is not required.

```

Patient* PatientQueue::getNextPatient()
{
    Node* current;
    if (head == NULL)
    {
        return NULL;
    }
    current = head;
    if (head == lastDisabledPatient)
    {
        lastDisabledPatient = NULL;
    }
    head = head->getNextNode();
    if (head == NULL)
    {
        tail = NULL;
    }
    else
    {
        head->setPreviousNode(NULL);
    }
    size--;
    return current->getPatient();
}

```

*Figure 11: Get Next Patient (Patient Queue)*

The get next patient function is used to retrieve the next patient from the waiting queue by the nurse. The function would display a message telling the user that the list is empty in the event that the head node is null (no person in the waiting queue). In the event that the list does contain a patient, then the function would check if the patient is the last disabled patient within the list, as disabled patients would be prioritized. In the event that it is the last disabled person, then the attribute that keeps track of the address node on the last disabled person would be set to null. In the event that the next address would be null, then the tail address would also be set to null as the list would now be empty. If there is a next patient, then the address of the previous node of the head address would now be set to null as the patient would now be the first on the waiting list. The function would then return the first patient within the waiting list.

```

int PatientQueue::removePatient(std::string patientID)
{
    Node* current;
    if ((head->getPatient()->getPatientID()) == patientID)
    {
        current = head;
        head = head->getNextNode();
        if (head != NULL)
            head->setPreviousNode(NULL);
        if (current == lastDisabledPatient)
        {
            lastDisabledPatient = NULL;
        }
        delete current;
        size--;
        return 1;
    }
    else
    {
        current = head->getNextNode();
        while (current != NULL)
        {
            if (current->getPatient()->getPatientID() == patientID)
            {
                current->getPreviousNode()->setNextNode(current->getNextNode());
                if (current->getNextNode() != NULL)
                    current->getNextNode()->setPreviousNode(current->getPreviousNode());
                if (current == lastDisabledPatient)
                {
                    lastDisabledPatient = current->getPreviousNode();
                }
                delete current;
                size--;
                return 1;
            }
            else
            {
                current = current->getNextNode();
            }
        }
        return 0;
    }
}

```

*Figure 12 Remove Patient Method*

The figure above displays the method used to remove patients from the PatientQueue linked list. The method accepts the patient ID as a reference to search for the patient to be removed. A new node pointer named “current” is then created to traverse the linked list. The function first checks if the patient referenced by the head node in the list has the matching IDs with the one passed in. If they match, the current will be equalled to the head, and the head will be



moved to its next node. The program will then check whether the head is equals NULL, implying that the list is empty if it is. If not, then the previousNode attribute of the head will be set as NULL. Then, the current node is then checked if it equals to the lastDisabledPatient node, if yes, then the lastDisabledPatient will be set as NULL, indicating that there are no disabled patients left in the queue, then the current node will be deleted. If the patient referenced in the head node does not have matching IDs, the current node will then be set as the next node. A loop is then initiated where the current node will traverse the entire linked list until it finds the node that points to a patient with the matching ID, or until it reaches the end of the linked list. If the patient is found, then the nextNode attribute of currents' previousNode of the current node will be set as currents' next node, and the previousNode attribute of currents' nextNode will be set as currents' previousNode. If the current node is also equalled to the lastDisabledPatient node, then the lastDisabledNode will be set as current nodes' previousNode. This is because disabled patients will be prioritised and entered at the front of the list, and if the deleted patient is the last disabled one and they are removed, it immediately indicates that the patient before them is the new last disabled patient in the queue. After the patient has been removed, the method will return 1 to indicate that the patient has successfully been removed, otherwise, 0 will be returned to indicate that the patient was not found.

```

Patient* PatientQueue::getNextPatient()
{
    Node* current;
    if (head == NULL)
    {
        return NULL;
    }
    current = head;
    if (head == lastDisabledPatient)
    {
        lastDisabledPatient = NULL;
    }
    head = head->getNextNode();
    if (head == NULL)
    {
        tail = NULL;
    }
    else
    {
        head->setPreviousNode(NULL);
    }
    size--;
    return current->getPatient();
}

```

*Figure 13 Get next Patient method*

The getNextPatient method is a method within the PatientQueue linked list, this method does not exist within the regular linked list class as this method is inherited in the PatientQueue class. The function would initialize a variable named current of type node and would validate if the head reference has any address, if the head node does not have an address, then the function would return NULL, otherwise, the current variable would be set equal to the head reference. The head node represents the first patient on top of the patient queue. The function would then check whether the head node is also the last disabled patient within the waiting list, otherwise, the head reference would be set to the next patient within the queue and check if the head reference is NULL, in the event that it is NULL (meaning no more patients) then the tail node would also be set to NULL. The size of the linked list would be decreased and return a patient object which is from the current node.

## Node Class

```
class Patient;

class Node {
    Node* previousNode;
    Patient* patient;
    Node* nextNode;

public :
    Node(Patient* patient);
    Node* getPreviousNode();
    void setPreviousNode(Node* newPreviousNode);
    Node* getNextNode();
    void setNextNode(Node* newNextNode);
    Patient* getPatient();
};
```

*Figure 14 Node class definition*

Figure 14 shows all of the members in the node class. The main function of the Node class is to encapsulate the patient object and have two pointers which point to its previous node and next node. The node class is important as it can be regarded as the building block for the linked list.

```
Node::Node(Patient* patient) {
    previousNode = NULL;
    nextNode = NULL;
    this->patient = patient;
}
```

*Figure 15 Node class constructor*

Figure 15 shows the constructor for the node class. The constructor will accept a pointer pointing towards a patient object and then assigns it to the patient field under the node class. It also initializes the previousNode pointer and nextNode pointer to null by default. This is

because assigning the previousNode and nextNode pointer to null minimizes bugs since it is assumed that the links will only be set to pointing other nodes during the insertion process.

```
Node* Node::getPreviousNode() {  
    return previousNode;  
}
```

*Figure 16 Get Previous Node method*

The getPreviousNode method is used to return the address of the node before the current one in the linked list. It is mainly used when traversing the linked list and having to reference a particular nodes' previous node.

```
void Node::setPreviousNode(Node* newPreviousNode) {  
    previousNode = newPreviousNode;  
}
```

*Figure 17 Set Previous Node method*

The setPreviousNode method above is used to define the previousNode address of a specific node. It is mainly used when adding a node into a linked list where the previous node address of the node added needs to be defined, or if a node is removed from the linked list, the previousNode attribute of the node next to the removed node will need to be re-defined using this method.

```
Node* Node::getNextNode() {  
    return nextNode;  
}
```

*Figure 18 Get Next Node method*

The getNextNode method is used to return the address of the node after the current one in the linked list. It is mainly used when traversing the linked list and having to reference a particular nodes' next node.

```
void Node::setNextNode(Node* newNextNode) {  
    nextNode = newNextNode;  
}
```

*Figure 19 Set Next Node method*

The setNextNode method above is used to define the nextNode address of a specific node. It is mainly used when adding a node into a linked list where the next node address of the node added needs to be defined, or if a node is removed from the linked list, the nextNode attribute of the node previous of the removed node will need to be re-defined using this method.

```
Patient* Node::getPatient() {  
    return patient;  
}
```

*Figure 20 Get Patient method*

The get patient method is used to return a pointer to a patient object from a node. This method is mainly utilised in searching functions where the patient object addresses are required to be retrieved from their respective nodes.

## Doctor Class

```
class PatientQueue;
class LinkedList;
class Patient;

class Doctor
{
    PatientQueue* patientQueue;
    LinkedList* historyList;
    std::string name;

public:
    Doctor(PatientQueue* waitingList, LinkedList* historyList);
    void displayDoctorMenu();
    void viewInfo();
    void modifyMedicalInformation(Patient* patient);
    void treatPatient();
    void searchPatient();
    void patientEditor(LinkedList* patientList);
    void setName(std::string name);
    std::string getName();
};
```

*Figure 21 System Overview for Doctor Class*

The doctor class is used to contain variables and functions that are relevant to or will be used by doctor staff in Klinik Sulaiman. There are 8 methods in total that integrate and implement functionalities from methods in the Utility and LinkedList class with end-user compatibility. Methods in the doctor class include viewing patient visit history, patient queue, as well as treating, modifying, and searching for patients.

```
void Doctor::modifyMedicineInformation(Patient* patient)
{
    std::string medicineInfo;
    char changeSicknessDescription;

    std::cout << "----- Current Patient Details -----" << std::endl;
    std::cout << patient->toString() << std::endl << std::endl;
    std::cout << "Enter the medicine information prescribed for this patient: ";
    getline(std::cin, medicineInfo);

    patient->setMedicineInformation(medicineInfo);

    std::cout << "\nEnter [1] if you wish to modify the sickness description of patient or any other key to terminate this operation: ";
    std::cin >> changeSicknessDescription;
    std::cin.ignore(256, '\n');

    if (changeSicknessDescription == '1')
    {
        std::string newSicknessDescription;
        std::cout << "\nEnter the sickness description for this patient: ";
        getline(std::cin, newSicknessDescription);
        patient->setSicknessDescription(newSicknessDescription);
    }

    std::cout << "\nAll modification saved successfully\n";
}
```

*Figure 22 Modify medicine information method*

Figure 22 shows the modify medicine information method under the doctor class. The modify medicine information method takes in a pointer pointing towards a patient object as a parameter and does not return any value. Then, the function will prompt the doctor to enter the patient medicine information and give the doctor a choice on whether to change the sickness description of the patient. Since the modify medicine information method takes in a pointer that is pointing to the patient object, hence the changes made to the patient inside this method will be reflected in the entire program. In addition, the setMedicineInformation and setSicknessDescription method are used here to change the medicine information and sickness description.

```

void Doctor::treatPatient() {
    LinkedList* allPatient = historyList->search(name, 5);
    LinkedList* treatPatient = allPatient->search("", 4);
    char confirmation = '1';

    std::cout << "There are " << treatPatient->getSize() << " patients that need your attention to enter their medicine information." << std::endl;
    if (treatPatient->getHeadReference() != NULL) {
        Node* currentPatient = treatPatient->getHeadReference();
        while (currentPatient != NULL && confirmation == '1') {
            std::cout << "\nWould you like to continue to enter patient medicine information now ? Enter [1] for yes or any other key for no :";
            std::cin >> confirmation;
            std::cin.ignore(256, '\n');
            if (confirmation == '1') {
                system("cls");
                modifyMedicineInformation(currentPatient->getPatient());
                currentPatient = currentPatient->getNextNode();
            }
            else
                continue;
        }
    }
    std::cout << "\nRedirecting you back to main menu..." << std::endl << std::endl;
    system("pause");
}

```

*Figure 23 Treat patient method*

Figure 23 shows the treat patient method under the doctor class. The treat patient method does not accept any parameter and does not return any value. This method primarily acts as an “interface” between the doctor user and the modify medicine information method as discussed in Figure 22. In this program, it is assumed that patient without any medicine information in the history list requires the doctor’s attention to enter their medical information before they are considered to be treated by the doctor. Therefore, this function filters patients in the history list based on the doctor’s name that is logged into the system and patients’ medical information status. This can be done by using two search functions as shown in the first two lines of code in Figure 23. The first search returns a linked list with all patients from history list with the same doctor name, and the second search is performed on the linked list returned from the first search, to search for all the remaining patients with empty medical information. Afterward, a while loop will be used to loop through the linked list which contains all the patients that require the doctor’s attention. If the linked list is not empty, the patient contained in the current node will be passed into the modify medicine information method at each iteration until every node in the linked list is traversed through, or the doctor prefers to exit the method.



```
void Doctor::displayDoctorMenu() {
    char choice = 0;
    while (true) {
        std::cout << "Doctor menu " << std::endl;
        std::cout << "1. View Patient History List and Patient Waiting List" << std::endl;
        std::cout << "2. Treat Patients" << std::endl;
        std::cout << "3. Search Patients" << std::endl;
        std::cout << "4. Logout" << std::endl;
        std::cout << "Select one of the options displayed above by entering the number : ";
        std::cin >> choice;
        std::cin.ignore(256, '\n');
        std::cout << std::endl;
        if (choice == '1')
            viewInfo();
        else if (choice == '2')
            treatPatient();
        else if (choice == '3')
            searchPatient();
        else if (choice == '4')
            break;
        else {
            std::cout << "\n\nWARNING: The input you have entered is not supported by the system, please select from the menu\n\n";
            system("pause");
        }
        system("cls");
    }
}
```

*Figure 24: Display Doctor Menu function*

The function above is for displaying the menu for doctors to interact with the system. There are 4 main options for the doctor to choose. The first option would call the viewInfo method which would direct the doctor to a separate sub-menu that shows a list of different viewing methods as well as viewing from different linked list as shown in the figure below. The second option is where the doctor would like to treat patients that has been called by the nurse and would call for the treatPatient method, this method would allow the doctor to modify records of the patient and be permanently saved into the history list when finished. The third option would be to search for specific patients which would call the searchPatient method. The searchPatient method would redirect the user to its sub-menu and would require the user to choose the between searching by patient id, their first name or sickness description.

```
void Doctor::viewInfo()
{
    int choice = 0;

    while (choice != 9)
    {
        system("cls");
        std::cout << "=== Doctor Viewing Menu ===\n";
        std::cout << "1. View entire patient waiting list\n";
        std::cout << "2. View patient waiting list in page by page mode\n";
        std::cout << "3. View entire patient visit history list (Descending order by visit time)\n";
        std::cout << "4. View patient visit history list in page by page mode\n";
        std::cout << "5. View sorted history list based on patient first name\n";
        std::cout << "6. View sorted waiting list based on patient first name\n";
        std::cout << "7. View sorted history list based on sickness description\n";
        std::cout << "8. View sorted waiting list based on sickness description\n";
        std::cout << "9. Go Back to previous menu\n";
        std::cout << "Enter a number above: ";

        std::cin >> choice;
        if (std::cin.fail())
        {
            std::cout << "\n\nWARNING: The input you have entered is not supported by the system, please select from the menu\n\n";
            system("pause");
            std::cin.clear();
            std::cin.ignore(256, '\n');
            continue;
        }
        std::cin.ignore(256, '\n');
        std::cout << std::endl;
    }
}
```

Figure 25 view info method (Part 1)

```
switch (choice)
{
case 1:
    if (patientQueue->getHeadReference() != NULL)
        patientQueue->displayList();
    else
        std::cout << "There are 0 patients in the waiting list." << std::endl;
    break;

case 2:
    if (patientQueue->getHeadReference() != NULL)
        Utility::viewPatient(patientQueue);
    else
        std::cout << "There are 0 patients in the waiting list." << std::endl;
    break;

case 3:
    if (historyList->getHeadReference() != NULL)
        historyList->displayList();
    else
        std::cout << "There are 0 patients in the history list." << std::endl;
    break;

case 4:
    if (historyList->getHeadReference() != NULL)
        Utility::viewPatient(historyList);
    else
        std::cout << "There are 0 patients in the history list." << std::endl;
    break;

case 5:
    if (historyList->getHeadReference() != NULL)
        Utility::mergeSort(historyList, 1)->displayList();
    else
        std::cout << "There are 0 patients in the history list." << std::endl;
    break;

case 6:
    if (patientQueue->getHeadReference() != NULL)
        Utility::mergeSort(patientQueue, 1)->displayList();
    else
        std::cout << "There are 0 patients in the history list." << std::endl;
    break;
```

Figure 26: view info method (Part 2)

```
case 7:
    if (patientQueue->getHeadReference() != NULL)
        Utility::mergeSort(historyList, 2)->displayList();
    else
        std::cout << "There are 0 patients in the history list." << std::endl;
    break;

case 8:
    if (patientQueue->getHeadReference() != NULL)
        Utility::mergeSort(patientQueue, 2)->displayList();
    else
        std::cout << "There are 0 patients in the history list." << std::endl;
    break;

case 9:
    break;

default:
    std::cout << "\n\nWARNING: The input you have entered is not supported by the system, please select from the menu\n";
}

std::cout << std::endl << std::endl;
system("pause");
}

return;
```

*Figure 27 view info method (Part 3)*

The view info function would redirect the user to its sub-menu and would only be displayed once the doctor has decided to view the history and waiting list (chose option 1) from the main menu. The first and second options are for viewing the patient list, just in different display formats. As shown in the figure above, before calling each individual printing methods, linked list validation would take place where in the event that a linked list does not have a head reference meaning it is empty, then the system would inform the user that there is no one within the linked list. The third and fourth options are for the history list, and is also to allow the doctor to have the flexibility in having different viewing modes. The display method would loop through the entire list and display it all at one while the viewPatient method which accepts an linked list as a parameter allows to user to traverse between the linked list in a back and forth motion.

```

void Doctor::searchPatient()
{
    int searchMode = 0;
    std::string cont;
    std::string searchReference;
    while (searchMode != 5)
    {
        system("cls");
        std::cout << "=== Doctor Searching Menu ===\n";
        std::cout << "1. Search by Patient ID \n2. Search by Patient First Name \n3. Search by Sickness Description" <<
            "\n4. Search by Medicine Information \n5. Go back\n\nPlease select an option to search for the patients' profile: ";
        std::cin >> searchMode;
        if (searchMode < 1 || searchMode > 5 || std::cin.fail())
        {
            std::cin.clear();
            std::cin.ignore(256, '\n');
            std::cout << "\n\nWARNING: The input you have entered is not supported by the system, please select from the menu.\n";
            std::cout << std::endl << std::endl;
            system("pause");
            continue;
        }
        std::cin.clear();
        std::cin.ignore(256, '\n');
        if (searchMode == 5)
        {
            continue;
        }
        else
        {
            std::cout << "\nPlease enter the patient details to be searched with: ";
            getline(std::cin, searchReference);
            LinkedList* resultsFromPatientQueue = patientQueue->search(searchReference, searchMode);
            LinkedList* resultsFromHistoryList = historyList->search(searchReference, searchMode);
            LinkedList* finalResults = LinkedList::concatLists(resultsFromPatientQueue, resultsFromHistoryList);
            if (finalResults->getHeadReference() == NULL) {
                std::cout << "\nPatient(s) not found!\n";
            }
            else {
                char modifyPatient = ' ';
                finalResults->displayList();
                std::cout << "\nWould you like to edit the information of any patient(s) displayed above ? " << std::endl;
                std::cout << "Enter [1] for yes or any other key for no : ";
                std::cin >> modifyPatient;
                std::cin.ignore(256, '\n');
                if (modifyPatient == '1') {
                    patientEditor(finalResults);
                }
            }
            std::cout << std::endl << std::endl;
            system("pause");
        }
        return;
    }
}

```

*Figure 28 Search Patient method*

The figure above shows the search patient method under the doctor class which searches for patients from both the waiting list as well as the visit history list. A loop is used in the entire function and will only stop when searchMode is 5, which is the “go back” option. Firstly, a menu with a set of options is displayed to the user, and they will be prompted for an input for the attribute used to search their patient with based on the menu. If the user enters an invalid

or non-numerical value, the program will display an error message, and the method will loop again. If the input is valid, the user will be prompted again to enter a string as a reference to search for their patient. After that, the search method is called and the search mode and reference are passed in as parameters. Then, 2 new LinkedList are initialised, one to store the results returned by searching the history list, and the other for the visit history results. Finally, another LinkedList is created to store the combined results from both the waiting list and visit history list. If the head node of the finalResults linked list is empty, then the system will display a message indicating that there are no patients found, otherwise, it will print all the details of the patients which matching details for attributes passed in by the user. Before ending the method, the program will ask the user if they intend to edit information of any of the users searched. If the user entered '1', the program would call the patientEditor method where they can modify their details. And if they entered any other input, the method ends, and the user will be re-directed back to the main menu.

```
void Doctor::patientEditor(LinkedList* patientList) {
    char attribute = ' ';
    char traverseDir = ' ';
    Node* currentNode = patientList->getHeadReference();
    while (currentNode != NULL) {
        system("cls");
        std::cout << "----- Patient Editor -----" << std::endl << std::endl;
        std::cout << currentNode->getPatient()->toString() << std::endl;
        std::cout << "Would you like to modify this patient's record ? ";
        std::cout << "Press : " << std::endl;
        if (currentNode->getPreviousNode() != NULL)
            std::cout << "[0] to move to the previous patient" << std::endl;
        if (currentNode->getNextNode() != NULL)
            std::cout << "[1] to move to the next patient" << std::endl;
        std::cout << "[2] to stay at current patient and modify other attributes " << std::endl;
        std::cout << "[3] to exit the patient editor" << std::endl;
        std::cin >> traverseDir;
        std::cin.ignore(256, '\n');
        if (traverseDir == '0' && currentNode->getPreviousNode() != NULL) {
            currentNode = currentNode->getPreviousNode();
        }
        else if (traverseDir == '1' && currentNode->getNextNode() != NULL) {
            currentNode = currentNode->getNextNode();
        }
        else if (traverseDir == '3')
            return;
    }
}
```

*Figure 29 Patient editor method (Part 1)*

Figure 29 shows part of the code snippet for the patient editor method.

```

else if (traverseDir == '2') {
    std::cout << "Press : " << std::endl;
    std::cout << "[0] to change patient's first name" << std::endl;
    std::cout << "[1] to change patient's last name" << std::endl;
    std::cout << "[2] to change patient's age " << std::endl;
    std::cout << "[3] to change patient's gender" << std::endl;
    std::cout << "[4] to change patient's phone number" << std::endl;
    std::cout << "[5] to change patient's address" << std::endl;
    std::cout << "[6] to change patient's sickness description" << std::endl;
    std::cout << "[7] to change patient's disability option" << std::endl;
    std::cout << "[8] to change patient's doctor name" << std::endl;
    std::cout << "[9] to change patient's medicine information" << std::endl;
    std::cout << "Any other key to go back" << std::endl;

    std::cin >> attribute;
    std::cin.ignore(256, '\n');
    if (attribute == '0' || attribute == '1' || attribute == '2' || attribute == '3' || attribute == '4' || attribute == '5'
        || attribute == '6' || attribute == '7' || attribute == '8' || attribute == '9') {
        currentNode->getPatient()->modifyRecord(attribute);
        std::cout << "All modifications are saved. Changes will be reflected on the next screen." << std::endl << std::endl;
    }
    system("pause");
}
else {
    std::cout << "Invalid operation. Please try again.";
    system("pause");
}
}
std::cout << "There are no remaining patients that can be modified...";
}

```

*Figure 30 Patient editor (Part 2)*

Figure 30 shows the last part of the code snippet for the patient editor method. The patient editor method takes in a linked list as a parameter and does not return any value. Essentially, the patient editor method will loop through the linked list and print out the patient details at each node. In addition, the patient editor method will also prompt the doctor on whether to edit the patient details. All the patient details are editable except for patient ID, patient's visit time, and patient's visit day. This is because the three attributes mentioned are generated by the system and the program does not provide any way to modify system-generated information to prevent any additional complexities in handling them after modification and can potentially introduce errors to the program.



## Nurse Class

```
class PatientQueue;
class LinkedList;
class Patient;

class Nurse {
    PatientQueue* patientQueue = NULL;
    LinkedList* historyList = NULL;
    std::string name;

public:
    Nurse(PatientQueue* patientQueue, LinkedList* historyList);
    void displayNurseMenu();
    void callPatient();
    Patient* createPatient();
    void viewWaitingList();
    void addPatient(Patient* newPatient);
    void deletePatient();
    void searchPatient();
    std::string getName();
    void setName(std::string name);
};
```

*Figure 31 System overview for Nurse class*

The nurse class consists of 2 linked lists and a “name” variable of type string to contain the name of the nurse when logging in. The class contains 10 methods in total. The nurse constructor would accept a linked list of type PatientQueue as well as another linked list which is the historyList. These two linked lists would be initialized and used throughout all the other methods within the class. All the methods within the class would be to display the menu as well as functions to host the task needed to be performed by the nurse.

```
□ Patient* Nurse::createPatient() {  
    std::string firstName;  
    std::string lastName;  
    std::string age;  
    std::string gender;  
    std::string phone;  
    std::string address;  
    std::string sicknessDescription;  
    std::string patientInformation;  
    std::string doctorName;  
    std::string medicineInformation = "";  
    std::string disabledStatus;  
    bool isDisabled;  
    bool validDoctor = false;  
    std::string patientID = "PID" + std::to_string(patientQueue->getSize() + historyList->getSize()+ 1);  
    system("cls");  
    std::cout << "-----Create patient menu-----\n\n";  
    std::cout << "Please enter the patient's first name: ";  
    getline(std::cin, firstName);  
    std::cout << "Please enter the patient's last name: ";  
    getline(std::cin, lastName);  
    std::cout << "Please enter the patient's age: ";  
    getline(std::cin, age);  
}
```

*Figure 32 Create patient method (part 1)*

```
while (!(Utility::stringNumber(age)) || age=="")
{
    std::cout << "\nInvalid input, please enter numeric values only.\n";
    std::cout << "Please enter the patient's age: ";
    getline(std::cin, age);
}
std::cout << "Please enter the patient's gender (Male/Female/Others): ";
getline(std::cin, gender);
std::transform(gender.begin(), gender.end(), gender.begin(), [](unsigned char c)
{ return std::tolower(c); });
while (gender != "male" && gender != "female" && gender != "others")
{
    std::cout << "\nInvalid input, please try again with options given.\n";
    std::cout << "Please enter the patient's gender (Male/Female/Others): ";
    getline(std::cin, gender);
    std::transform(gender.begin(), gender.end(), gender.begin(), [](unsigned char c)
    { return std::tolower(c); });
}
std::cout << "Please enter the patient's phone number: ";
getline(std::cin, phone);
while (!(Utility::stringNumber(phone)) || phone=="")
{
    std::cout << "\nInvalid input, please enter numeric values for phone numbers only.\n";
    std::cout << "Please enter the patient's phone number: ";
    getline(std::cin, phone);
}
std::cout << "Please enter the patient's address: ";
getline(std::cin, address);
std::cout << "Please enter the patient's sickness description: ";
getline(std::cin, sicknessDescription);
std::cout << "Is the patient disabled? (Yes/No): ";
getline(std::cin, disabledStatus);
std::transform(disabledStatus.begin(), disabledStatus.end(), disabledStatus.begin(), [](unsigned char c)
{ return std::tolower(c); });
while (disabledStatus != "yes" && disabledStatus != "no")
{
    std::cout << "\nInvalid input, please try again.\n";
    std::cout << "Is the patient disabled? (Yes/No): ";
    getline(std::cin, disabledStatus);
    std::transform(disabledStatus.begin(), disabledStatus.end(), disabledStatus.begin(), [](unsigned char c)
    { return std::tolower(c); });
}
```

Figure 33 Create patient method (Part 2)

```

if (disabledStatus == "yes")
{
    isDisabled = true;
}
else
{
    isDisabled = false;
}
int numberOfDoctors=0;
std::string* availableDoctors = Utility::getDoctors(&numberOfDoctors);
std::cout << "\nAvailable doctors:" << std::endl;
for (int i = 0; i < numberOfDoctors;i++)
{
    std::cout << " - " << availableDoctors[i] << std::endl;;
}
std::cout << "\nPlease enter the patient's doctor name from above: ";
getline(std::cin, doctorName);
while (validDoctor==false)
{
    for (int i = 0; i < numberOfDoctors;i++)
    {
        if (doctorName == availableDoctors[i])
        {
            validDoctor = true;
            break;
        }
    }
    if (validDoctor==false)
    {
        std::cout << "\nInvalid input, please select a doctor from the list above!";
        std::cout << "\nPlease enter the patient's doctor name from above: ";
        getline(std::cin, doctorName);
    }
}
Patient* newPatient = new Patient(patientID, firstName, lastName, age, gender, phone, address, sicknessDescription, medicineInformation, doctorName, isDisabled);
return newPatient;

```

*Figure 34 Create Patient method (Part 3)*

The createPatient method consist of all the necessary attributes which is needed when registering a new patient while two Boolean variables named isDisabled and validDoctor would be set to NULL and false by the default. When the wishes to register a new patient into the patientQueue, the system would call for the createPatient method and would prompt the user to enter all the necessary data. When the nurse is entering the age of a patient, a number of validation processes would take place such as making sure no alphabets or empty inputs are made and would prompt the user to re-enter the age data if it does happen. The same validation process would also be carried out for gender, the system only accepts inputs such as male, female and others. The system would automatically convert all inputs from gender to lower case thus the system input is not case sensitive. The nurse would also need to state whether the patient is disabled or not, this input is very important as disabled patient would place the user ahead in the queue before the non-disabled patients. The program would later ask the nurse for which doctor would be assigned for the patient, here input validation would also be taken place where is the nurse enters a non-existing doctor, the system would reject the input and ask for a proper input again. Once all the information has been gathered, the method would then call for the patient constructor and pass all the gather data into the patient constructor to create a new

patient. Once the new patient has been created, the function would then return the new patient object.

```

void Nurse::searchPatient()
{
    int searchMode = 0;
    std::string cont;
    std::string searchReference;
    while (searchMode != 3)
    {
        system("cls");
        std::cout << "=== Nurse Searching Menu ===\n";
        std::cout << "1. Search by Patient ID \n2. Search by Patient First Name\n3.Go Back" <<
            "\n\nPlease select an option to search for the patients' profile: ";
        std::cin >> searchMode;
        if (searchMode < 1 || searchMode>3 || std::cin.fail())
        {
            std::cin.clear();
            std::cin.ignore(256, '\n');
            std::cout << "\n\n WARNING: The input you have entered is not supported by the system, please select from the menu.\n";
            std::cout << std::endl << std::endl;
            system("pause");
            continue;
        }
        std::cin.clear();
        std::cin.ignore(256, '\n');
        if (searchMode == 3)
        {
            continue;
        }
        else
        {
            std::cout << "\nPlease enter the patient details to be searched with: ";
            getline(std::cin, searchReference);
            LinkedList* patientList = patientQueue->search(searchReference, searchMode);
            if (patientList->getHeadReference() != NULL)
            {
                std::cout << std::endl;
                patientList->displayList();
            }
            else
            {
                std::cout << "\nPatient not found!\n";
            }
        }
        std::cout << std::endl << std::endl;
        system("pause");
    }
    return;
}

```

*Figure 35 search patient method*

The searchPatient method is used to display the search menu and prompts the nurse for additional inputs needed to perform the searching process. Once this method has been called, this method would display a sub-menu asking for which searching mode the user would like to use. The system would only support inputs from 1-3, if other inputs are not numeric or not in the range of 1-3, it would show an error message and ask the nurse to re-enter again, once the nurse has successfully chosen the searchMode, the nurse would need to enter the searchReference. The variable named patientList of type linked list would be created to host the linked list that is returned from the search Method which requires the searchReference and searchMode as its parameters. If the returned linked list does not have a head reference, it will

inform the user that there are no patient records found, otherwise, it would print out the search results which are the patient details in the linked list returned.

```
void Nurse::viewWaitingList() {  
    if (patientQueue->getHeadReference() == NULL)  
    {  
        std::cout << "\nThere are 0 patients in the waiting list.\n\n";  
    }  
    else  
    {  
        std::cout << "\nHere are all the patients in the waiting list: \n\n";  
        patientQueue->displayList();  
    }  
    return;  
}
```

*Figure 36 view waiting list method*

The viewWaitingList method is a method that allows the nurse to view the entire patientQueue linked list. In the event that the head reference is NULL (meaning no nodes in the linked list) it would tell the user that there are 0 patients in the waiting list, otherwise, it would call the displayList method from the patientQueue which is a method native to the linked list class and prints all the nodes in a single sitting.



```
void Nurse::displayNurseMenu()
{
    int choice = 0;
    int viewChoice = 0;
    std::string confirm;
    while (choice != 8)
    {
        system("cls");
        std::cout << "=== Nurse Menu ===\n";
        std::cout << "1. View patient waiting list\n";
        std::cout << "2. Add patient to the waiting list\n";
        std::cout << "3. Call next patient for doctor visit\n";
        std::cout << "4. Search patient from waiting list\n";
        std::cout << "5. View patient waiting list sorted by time\n";
        std::cout << "6. View patient waiting list sorted by patient ID\n";
        std::cout << "7. Remove patient from waiting list\n";
        std::cout << "8. Logout\n\n";
        std::cout << "Enter a number above: ";

        std::cin >> choice;
        std::cin.clear();
        std::cin.ignore(256, '\n');
        if (std::cin.fail())
        {
            std::cin.clear();
            std::cin.ignore(256, '\n');
            continue;
        }
    }
}
```

*Figure 37 Display Nurse Menu (Part 1)*

Figure 37 shows part of the code snippet for the display nurse menu method under the nurse class.

```
switch (choice)
{
case 1:
    std::cout << "Please enter [1] to view full waiting list, and [2] to view via page-by-page: ";
    std::cin >> viewChoice;
    if (std::cin.fail() || (viewChoice != 1 && viewChoice != 2))
    {
        std::cin.clear();
        std::cin.ignore(256, '\n');
        std::cout << "\n\nWARNING: The input you have entered is not supported by the system, please select from the menu\n";
    }
    else
    {
        std::cin.clear();
        std::cin.ignore(256, '\n');
        if (viewChoice == 1)
            viewWaitingList();
        else if (viewChoice == 2)
            Utility::viewPatient(patientQueue);
    }
    break;
case 2: {
    Patient* newPatient = createPatient();
    std::cout << "\n---Patient Details---\n";
    std::cout << newPatient->toString();
    std::cout << "\nConfirm adding patient into the waiting list? (Yes/No): ";
    getline(std::cin, confirm);
    std::transform(confirm.begin(), confirm.end(), confirm.begin(), [](unsigned char c)
    { return std::tolower(c); });
    while (confirm != "yes" && confirm != "no")
    {
        std::cout << "\nInvalid input, please enter 'Yes' or 'No' to confirm for adding patient: ";
        getline(std::cin, confirm);
        std::transform(confirm.begin(), confirm.end(), confirm.begin(), [](unsigned char c)
        { return std::tolower(c); });
    }
    if (confirm == "no")
    {
        std::cout << "\nProcess has been cancelled.\n";
        break;
    }
    else
    {
        addPatient(newPatient);
    }
}
break;
```

Figure 38 Display nurse menu (Part 2)

Figure 38 shows part of the code snippet for the display nurse menu method under the nurse class.

```
case 3:
    callPatient();
    break;

case 4:
    searchPatient();
    break;

case 5: {
    LinkedList* visitTimeSorted = Utility::mergeSort(patientQueue, 3);
    if (visitTimeSorted->getHeadReference() != NULL)
        visitTimeSorted->displayList();
    else
        std::cout << "\nThere are 0 patients in the patient queue" << std::endl;
}
    break;

case 6: {
    LinkedList* patientIDSorted = Utility::mergeSort(patientQueue, 0);
    if (patientIDSorted->getHeadReference() != NULL)
        patientIDSorted->displayList();
    else
        std::cout << "\nThere are 0 patients in the patient queue" << std::endl;
}
    break;

case 7:
    deletePatient();
    break;

case 8:
    return;

default:
    std::cout << "\n\nWARNING: The input you have entered is not supported by the system, please select from the menu\n";
}
std::cout << std::endl << std::endl;
system("pause");
}
return;
```

*Figure 39 Display nurse menu (Part 3)*

Figure 39 shows the last part of the code snippet for the display nurse menu method under the nurse class. Based on Figure 37, Figure 38 and Figure 39, the display nurse menu method does not take in any parameter and does not return any value. The main objective of the display nurse menu is to display the menu relevant to the nurse users and prompt the nurse users to select an operation to be performed by the program. Once an option is selected, the display nurse menu will call the relevant methods that handle the selected operation. The nurse users can also choose to log out from the program once they are done with all the operations.

```
void Nurse::addPatient(Patient* newPatient) {  
    std::cout << "\nInserting patient into the waiting list..." << std::endl << std::endl;  
    patientQueue->insertPatient(newPatient);  
    std::cout << "Patient added successfully" << std::endl;  
}
```

*Figure 40 Add patient method*

Figure 40 shows the add patient method under the nurse class. The add patient method acts as an interface between the insert patient method in the patient queue class and the nurse user. The add patient method takes in a pointer pointing towards the new patient object and does not return any value. The add patient method then proceeds to call the insert patient method to insert the new patient into the patient queue and notify the nurse user once the patient is inserted.

```
void Nurse::callPatient() {  
    Patient* currentPatient = patientQueue->getNextPatient();  
    if (currentPatient != NULL) {  
        std::cout << "\n-----Details of patient called-----\n\n";  
        std::cout << currentPatient->toString() << std::endl << std::endl;  
        historyList->insertAtFront(currentPatient);  
    }  
  
    else {  
        std::cout << "There are 0 patients in the waiting list right now." << std::endl;  
    }  
}
```

*Figure 41 Call patient method*

Figure 41 shows the call patient method under the nurse class. The call patient method acts as an interface between the get next patient method in the patient queue class and the nurse user. Through this, it hides away the complex details needed to retrieve the first patient from the patient queue. If there are no patients in the patient queue for the nurse to call, then this method will notify the nurse that the patient queue is empty, otherwise, it will print out the patient details and insert the patient into the history list. This is because the program assumes that once the patient is called, the patient will be meeting the doctor immediately after, therefore the patient is considered to have visited the doctor.

```
void Nurse::deletePatient() {  
    std::string patientIDDelete;  
    std::cout << "Enter ID of patient to be deleted : ";  
    getline(std::cin, patientIDDelete);  
    int result = patientQueue->removePatient(patientIDDelete);  
    if (result == 1)  
        std::cout << "Patient successfully deleted. " << std::endl;  
    else  
        std::cout << "Invalid Patient ID or Patient with this ID does not exist..." << std::endl;  
}
```

*Figure 42 Delete patient method*

Figure 42 shows the delete patient method under the nurse class. The delete patient method is self-explanatory, the method is responsible for deleting patients in the patient queue. It is also important to note that deletion is only possible for patient queue and not for history list to prevent any accidental deletion that can cause data loss. In addition, the nurse can only delete patients from the patient waiting list based on patient ID. Since patient ID is unique for each patient, this reduces any ambiguity when deleting patients. The delete patient method is included because some patients might leave the patient queue due to other reasons, which they are assumed to inform the nurses before leaving.

```
std::string Nurse::getName() {  
    return name;  
}
```

*Figure 43 Get nurse name method*

Figure 43 shows the get name method under the nurse class. It returns the name of the nurse as a string value.

```
void Nurse::setName(std::string name) {  
    this->name = name;  
}
```

*Figure 44 set name method*

The setName method would accept a string value, the string value that is passed into the function would set the name of the nurse within the nurse class.

## Patient Class

```
class Patient {  
    std::string patientID;  
    std::string firstName;  
    std::string lastName;  
    std::string age;  
    std::string gender;  
    std::string address;  
    std::string phone;  
    std::string sicknessDescription;  
    std::string medicineInformation;  
    std::string doctorName;  
    bool disabled;  
    int visitDay;  
    int visitMonth;  
    int visitYear;  
    int visitHour;  
    int visitMinute;  
    int visitSecond;  
  
public :  
  
    Patient (std::string patientID, std::string firstName, std::string lastName,  
            std::string age, std::string gender, std::string phone,  
            std::string address, std::string sicknessDescription,  
            std::string medicineInformation, std::string doctorName, bool disabled);  
  
    bool isDisabled();  
    std::string toString();  
    void setTime(int hour, int minute, int second);  
    void modifyRecord(char mode);  
    //Getters  
    std::string getPatientID();  
  
    std::string getFirstName();  
  
    std::string getSicknessDescription();  
    std::string getMedicineInformation();  
    std::string getDoctorName();  
  
    int getVisitHour();  
    int getVisitMinute();  
    int getVisitSecond();  
  
    //Setters  
    void setPatientID(std::string patientID);  
  
    void setFirstName(std::string firstName);  
  
    void setSicknessDescription(std::string sicknessDescription);  
    void setMedicineInformation(std::string medicineInformation);  
  
    void setDoctorName(std::string doctorName);  
  
    void setVisitDay(int Day, int Month, int Year);  
};
```

Figure 45 System Overview of Patient class

The patient class is mainly used to store the many attributes or variables that are required when registering a patient, such as patient ID, name, sickness, and so on. This class consists of many methods, but most of them are just getters and setters for variables within the class for encapsulation purposes.

```
bool Patient::isDisabled() {  
    return disabled;  
}
```

*Figure 46 Is Disabled method*

The method above is used to return the disabled attribute of the patient, it returns a Boolean expression based on whether or not the patient is disabled. This method is used when inserting a patient into the waiting queue as the program has to check if the patient is disabled or not to decide to place the patient at the start or end of the queue.

```
std::string Patient::toString()  
{  
    std::string patientProfile = "Patient ID: " + patientID + "\n" + "Patient first Name: "  
        + firstName + "\n" + "Patient last Name: " + lastName + "\n" + "Patient Age: "  
        + age + "\n" + "Patient Gender: " + gender + "\n" + "Patient Contact: " + phone + "\n" +  
        "Address : " + address + "\n" + "Responsible Doctor: " + doctorName + "\n"  
        + "Sickness Description: " + sicknessDescription + "\n" + "Medicine Information: "  
        + medicineInformation + "\n" + "Disabled : " + std::to_string(disabled)  
        + "\nVisiting Day(dd/mm/yy) : " + std::to_string(visitDay) + "/" + std::to_string(visitMonth)  
        + "/" + std::to_string(visitYear)  
        + "\nVisiting Time(hour / min / sec) : ";  
  
    if (visitHour <= 9)  
        patientProfile += "0";  
    patientProfile += std::to_string(visitHour) + ":";  
    if (visitMinute <= 9)  
        patientProfile += "0";  
    patientProfile += std::to_string(visitMinute) + ":";  
    if (visitSecond <= 9)  
        patientProfile += "0";  
    patientProfile += std::to_string(visitSecond) + "\n";  
  
    return patientProfile;  
}
```

*Figure 47 Patient to String method*

The figure above shows the toString method used to display the variables stored in the patient object. The method is used when searching the patients or calling the patient for their doctor visitation where their details will be displayed for the nurse or doctor to see. The method



returns a string that has a properly formatted output for the end-user display that can be easily integrated into other methods. In addition, several if statements were used for formatting the visiting time of the patient. If the hour, minute, or second of the visit time is less than 10, the toString method will add an additional '0' to the front. This is so that the output time would look like "09:30:05" instead of "9:30:5".

```

void Patient::modifyRecord(char mode) {
    if (mode == '0') {
        std::cout << "Enter the patient's new first name : ";
        getline(std::cin, firstName);
    }
    else if (mode == '1') {
        std::cout << "Enter the patient's new last name : ";
        getline(std::cin, lastName);
    }
    else if (mode == '2') {
        std::cout << "Enter the patient's new age : ";
        getline(std::cin, age);
        while (!(Utility::stringNumber(age)) || age == "")
        {
            std::cout << "\nInvalid input, please enter numeric values only.\n";
            std::cout << "Please enter the patient's new age: ";
            getline(std::cin, age);
        }
    }
    else if (mode == '3') {
        std::cout << "Enter the patient's gender (Male/Female/Others) : ";
        getline(std::cin, gender);
        std::transform(gender.begin(), gender.end(), gender.begin(), [](unsigned char c)
        { return std::tolower(c); });
        while (gender != "male" && gender != "female" && gender != "others")
        {
            std::cout << "\nInvalid input, please try again with options given.\n";
            std::cout << "Please enter the patient's gender (Male/Female/Others): ";
            getline(std::cin, gender);
            std::transform(gender.begin(), gender.end(), gender.begin(), [](unsigned char c)
            { return std::tolower(c); });
        }
    }
    else if (mode == '4') {
        std::cout << "Enter the patient's new phone number : ";
        getline(std::cin, phone);
        while (!(Utility::stringNumber(phone)) || phone == "")
        {
            std::cout << "\nInvalid input, please enter numeric values for phone numbers only.\n";
            std::cout << "Please enter the patient's new phone number: ";
            getline(std::cin, phone);
        }
    }
    else if (mode == '5') {
        std::cout << "Enter the patient's new address : ";
        getline(std::cin, address);
    }
    else if (mode == '6') {
        std::cout << "Enter the patient's new sickness description : ";
        getline(std::cin, sicknessDescription);
    }
}

```

Figure 48 Patient Modify Record method (Part 1)

```

else if (mode == '7') {
    std::cout << "Is the patient disabled ? (Yes/No): ";
    std::string disabledStatus;
    getline(std::cin, disabledStatus);
    std::transform(disabledStatus.begin(), disabledStatus.end(), disabledStatus.begin(), [](unsigned char c)
    { return std::tolower(c); });
    while (disabledStatus != "yes" && disabledStatus != "no")
    {
        std::cout << "\nInvalid input, please try again.\n";
        std::cout << "Is the patient disabled? (Yes/No): ";
        getline(std::cin, disabledStatus);
        std::transform(disabledStatus.begin(), disabledStatus.end(), disabledStatus.begin(), [](unsigned char c)
        { return std::tolower(c); });
    }
    if (disabledStatus == "yes")
    {
        disabled = true;
    }
    else
    {
        disabled = false;
    }
}
else if (mode == '8') {
    std::cout << "Enter the patient's new doctor name : ";
    int numberOfDoctors = 0;
    bool validDoctor = false;
    std::string* availableDoctors = Utility::getDoctors(&numberOfDoctors);
    std::cout << "\nAvailable doctors:" << std::endl;
    for (int i = 0; i < numberOfDoctors; i++)
    {
        std::cout << " - " << availableDoctors[i] << std::endl;
    }
    std::cout << "\nPlease enter the patient's doctor name from above: ";
    getline(std::cin, doctorName);
    while (validDoctor == false)
    {
        for (int i = 0; i < numberOfDoctors; i++)
        {
            if (doctorName == availableDoctors[i])
            {
                validDoctor = true;
                break;
            }
        }
        if (validDoctor == false)
        {
            std::cout << "\nInvalid input, please select a doctor from the list above!";
            std::cout << "\nPlease enter the patient's new doctor name from above: ";
            getline(std::cin, doctorName);
        }
    }
}
else if (mode == '9') {
    std::cout << "Enter the patient's new medicine information : ";
    getline(std::cin, medicineInformation);
}
}

```

*Figure 49 Patient Modify Record method (Part 2)*

The figure above displays the modify record method that is used by the doctor to change certain details of a patients' record. The method accepts a character variable as its parameter, and it will determine the attribute of which the doctor will be modifying. For instance, if '1' is passed in, the doctor will modify the patients' first name, and if '3' is passed, the patients' age will be modified, and so on. This method is also somewhat similar to that of the add patient method in the nurse class in the way that certain attribute input has input validations. For example, the "age" attribute will only accept numerical input, and the "doctorName" field only accepts

names of doctors who are currently working in the clinic, otherwise, and an error message will be displayed, indicating that the user input is invalid before prompting for another input.

```
//Getters
std::string Patient::getPatientID() {
    return patientID;
}

std::string Patient::getFirstName() {
    return firstName;
}

std::string Patient::getSicknessDescription() {
    return sicknessDescription;
}

std::string Patient::getMedicineInformation() {
    return medicineInformation;
}

std::string Patient::getDoctorName() {
    return doctorName;
}

int Patient::getVisitHour() {
    return visitHour;
}

int Patient::getVisitMinute() {
    return visitMinute;
}

int Patient::getVisitSecond() {
    return visitSecond;
}
```

*Figure 50 Patient Getters method*

The figure above shows all the getters methods used to retrieve all the attributes of patients within the patient class. These methods are primarily utilised by the searching function in the LinkedList class where the search reference passed in by the user has to be compared to the chosen attribute of the patients. Also, these methods are required when sorting the linked lists where attributes of the patients need to be accessed in other classes.

```
//Setters
void Patient::setPatientID(std::string patientID) {
    this->patientID = patientID;
}

void Patient::setFirstName(std::string firstName) {
    this->firstName = firstName;
}

void Patient::setSicknessDescription(std::string sicknessDescription) {
    this->sicknessDescription = sicknessDescription;
}

void Patient::setMedicineInformation(std::string medicineInformation) {
    this->medicineInformation = medicineInformation;
}

void Patient::setDoctorName(std::string doctorName) {
    this->doctorName = doctorName;
}

void Patient::setTime(int hour, int minute, int second) {
    visitHour = hour;
    visitMinute = minute;
    visitSecond = second;
}

void Patient::setVisitDay(int day, int month, int year) {
    this->visitDay = day;
    this->visitMonth = 1+month;
    this->visitYear = 1900+year;
}
```

*Figure 51 Patient Setters method*

The figure shows all the setter's method in the patient class. These methods are used in external classes when needing to modify the values of certain variables in the patient class. For instance, after treating a patient, the doctor will have to use the setMedicineInformation to add the information of the medicine prescribed to the patient after their visit. As for the set visit day method, the parameter year represents years since 1900, therefore to get the actual year, 1900 is added in. In addition, the parameter month represent months since January, hence to get the actual month, 1 is added in.

## Utility Class

```
class LinkedList;
class Doctor;
class Nurse;

class Utility {
public :
    static LinkedList* mergeSort(LinkedList* linkedList, int searchMode);
    static std::string getPassword();
    static void viewPatient(LinkedList* linkedList);
    static int login(std::string userName, std::string password, char isDoctor, Doctor* doctor, Nurse* nurse);
    static bool stringNumber(const std::string str);
    static std::string* getDoctors(int* numberOfDoctors);
};
```

*Figure 52 System Overview of Utility class*

The figure above shows the methods in the Utility class. The use of this class is to contain static methods that provide functionalities that will be commonly used by methods in other classes and can be called at any time without creating an object of the Utility class.

```
int Utility::login(std::string userName, std::string password, char isDoctor, Doctor* doctor, Nurse* nurse)
{
    bool found = false;
    int rowSize = 0;

    if (isDoctor == '1')
    {
        rowSize = sizeof doctorCredentials / sizeof doctorCredentials[0];

        for (int i = 0; i < rowSize; i++)
        {
            if (doctorCredentials[i][0] == userName && doctorCredentials[i][1] == password)
            {
                found = true;
                doctor->setName(userName);
                return 0;
            }
        }
    }
    else
    {
        rowSize = sizeof nurseCredentials / sizeof nurseCredentials[0];

        for (int i = 0; i < rowSize; i++)
        {
            if (nurseCredentials[i][0] == userName && nurseCredentials[i][1] == password)
            {
                found = true;
                nurse->setName(userName);
                return 1;
            }
        }
    }

    std::cout << "Sorry, but there is no matching records found\n";
    return -1;
}
```

*Figure 53 Login*

The login function is used at the beginning of the system by both the doctor and the nurse. The method would first initialize a Boolean variable named found and set its default value to false, while an integer variable named rowSize would be set to 0. If the user enters one when asking if the user is a doctor, the system would then calculate the rowSize of the doctor array which stores all the login credentials of doctors and loops through the array in a for loop to search for the matching credentials between the user input and the credentials within the doctor array. When asked if the user is a doctor and the user enters inputs other than 1, then it would perform the same process but with the nurse array. In the event that no matching records are found, the system would inform the user that there are no matching records, otherwise, the system would return a 0 or 1 depending on if the user is a doctor or nurse.

```
bool Utility::stringNumber(std::string str)
{
    for (char &c : str)
    {
        if (std::isdigit(c) == 0)
            return false;
    }
    return true;
}
```

*Figure 54 string number method*

The figure above showed the string number method in the Utility class. The method accepts a string as its parameter, where the function will create a loop to check for every character in the string and whether or not it is a numerical value. If there is a single character in the string that is not a digit, the method will return false, otherwise, it will return true. This method is used to ensure that the patients' phone number field entered by the nurse only contains numbers.



```

void Utility::viewPatient(LinkedList* linkedList)
{
    bool terminate = false;
    Node* currentNode = linkedList->getHeadReference();

    if (currentNode != NULL) {
        while (!terminate)
        {
            system("cls");
            std::cout << currentNode->getPatient()->toString();

            bool traverseNode = false;
            int choice = 0;
            std::cout << "Press : " << std::endl;
            if (currentNode->getPreviousNode() != NULL)
                std::cout << "[0] to view previous page " << std::endl;
            if (currentNode->getNextNode() != NULL)
                std::cout << "[1] to view next page " << std::endl;
            std::cout << "[2] to exit viewing: " << std::endl;
            std::cin >> choice;
            if (std::cin.fail()) {
                std::cin.clear();
                std::cin.ignore(256, '\n');
                std::cout << "\nSorry, but we do not support this input, please try again.\n\n";
                system("pause");
                continue;
            }
        }
    }
}

```

Figure 55 View patient method (Part 1)

```

std::cin.ignore(256, '\n');
switch (choice) {
case 0:
    if (currentNode->getPreviousNode() != NULL) {
        currentNode = currentNode->getPreviousNode();
        break;
    }
    else {
        std::cout << "\nSorry, but we do not support this input, please try again.\n\n";
        system("pause");
        break;
    }
case 1:
    if (currentNode->getNextNode() != NULL) {
        currentNode = currentNode->getNextNode();
        break;
    }
    else {
        std::cout << "\nSorry, but we do not support this input, please try again.\n\n";
        system("pause");
        break;
    }
case 2:
    terminate = true;
    break;
default :
    std::cout << "\nSorry, but we do not support this input, please try again.\n\n";
    system("pause");
}
}

```

Figure 56 view patient method (Part 2)

```
else
    std::cout << "\nThere are 0 patients in the waiting list.";
}
```

*Figure 57 view patient method (Part 3)*

The viewPatient method accepts a linked list as an argument. The main purpose of this function is to provide a different viewing method for the user, this method allows the user to traverse back and forth in the linked list and prints the data within each node, allowing the user to view each patients' individual records one at a time. The method would first initialize a Boolean variable called terminate and would be set to false by default. The currentNode variable of type node would be set to the value of the head reference of the linked list that has been parsed into the method. The method would first validate if the linked list contains any nodes, if there are none, then the system would inform the user that there are 0 patients within the linked list, otherwise it would enter a while loop, and prints out the currentNode, which was already initialized previously by the head reference. Once the patient records have been printed, the system would evaluate which user interface options would be displayed and made available based on the currentNode position. If the currentNode position is in the head node of the linked list, which means it would not have a previous node since it's the first, thus the interface would only show the ability to exit viewing or move to the next node. The same would also apply when the currentNode reaches the tail node of the linked list. However now would only be showing the options to exit viewing or go back to the previous node. If the currentNode is in between the head and tail node, the system would print out all the available traversal options on the interface. If the user enters an input that is not shown on the available options, the system would print out an error message telling the user that the option is not supported. When the user wishes to exit viewing, the system would set the terminate variable to true, and upon the next hydration of the while loop, would not fulfil the condition, thus terminating the while loop.

```
std::string Utility::getPassword() {  
    std::string password;  
    char lastChar = ' '  
    while (lastChar != 13) {  
        lastChar = _getch();  
        if (lastChar != 13 && lastChar != '\b') {  
            std::cout << '*';  
            password += lastChar;  
        }  
        else if (lastChar == '\b' && password.length() > 0) {  
            std::cout << "\b \b";  
            password.pop_back();  
        }  
    }  
    return password;  
}
```

*Figure 58 Get password method*

Figure 58 shows the get password method under the utility class. The get password method can be considered as one of the additional features included in the system. It is used to mask the input entered by the users with “\*”. Typically, this method is used when the users need to enter their password when logging into the system. The get password method does not accept any parameters and returns a string that contains the character input by the users. The \_getch() method provided in the “conio” library accepts characters entered by the users but does not display it as an output. Hence, manual checking is performed to detect whether the last character input by the user is the “enter key”, which has an ASCII value of 13. While the last character input is not equal to 13 (enter key), user input will continue to be accepted. If the users type in a backspace ('\b'), then the last character before the backspace key is removed from the password string and the last \* displayed on the screen will also be deleted by replacing it with a space. Otherwise, for each character input by the users, the command line will print out ‘\*’.

```

LinkedList* Utility::mergeSort(LinkedList* linkedList, int sortMode) {
    if (sortMode != 0 && sortMode != 1 && sortMode != 2 && sortMode != 3)
        return NULL;

    LinkedList* sortedList = linkedList;
    int segmentSize = 1;
    while (true) {
        Node* firstPart = sortedList->getHeadReference();
        Node* secondPart = firstPart;
        sortedList = new LinkedList();
        int totalMerges = 0;
        int firstPartSize;
        int secondPartSize;
        while (firstPart != NULL) {
            totalMerges++;
            firstPartSize = 0;
            secondPartSize = segmentSize;
            // Set two pointers to correct starting position
            while (firstPartSize < segmentSize && secondPart != NULL) {
                firstPartSize++;
                secondPart = secondPart->getNextNode();
            }
        }
    }
}

```

*Figure 59 Merge sort method (Part 1)*

Figure 59 shows one part of the code snippet for the merge sort method under the Utility class.

```

while (firstPartSize > 0 && (secondPartSize > 0 && secondPart != NULL)) {
    if (sortMode == 0) { // Compare patient ID
        if (firstPart->getPatient()->getPatientID().length() < secondPart->getPatient()->getPatientID().length()) {
            sortedList->append(firstPart->getPatient());
            firstPart = firstPart->getNextNode();
            firstPartSize--;
        }
        else if (firstPart->getPatient()->getPatientID().length() > secondPart->getPatient()->getPatientID().length()) {
            sortedList->append(secondPart->getPatient());
            secondPart = secondPart->getNextNode();
            secondPartSize--;
        }
        else {
            if (firstPart->getPatient()->getPatientID() <= secondPart->getPatient()->getPatientID()) {
                sortedList->append(firstPart->getPatient());
                firstPart = firstPart->getNextNode();
                firstPartSize--;
            }
            else {
                sortedList->append(secondPart->getPatient());
                secondPart = secondPart->getNextNode();
                secondPartSize--;
            }
        }
    }
}
}

```

*Figure 60 Merge sort method (Part 2)*

Figure 60 shows one part of the code snippet for the merge sort method under the Utility class.

```

else if (sortMode == 1) { // Compare patient first name
    if (firstPart->getPatient()->getFirstName() <= secondPart->getPatient()->getFirstName()) {
        sortedList->append(firstPart->getPatient());
        firstPart = firstPart->getNextNode();
        firstPartSize--;
    }
    else {
        sortedList->append(secondPart->getPatient());
        secondPart = secondPart->getNextNode();
        secondPartSize--;
    }
}
else if (sortMode == 2) { // Compare patient sickness description
    if (firstPart->getPatient()->getSicknessDescription() <= secondPart->getPatient()->getSicknessDescription()) {
        sortedList->append(firstPart->getPatient());
        firstPart = firstPart->getNextNode();
        firstPartSize--;
    }
    else {
        sortedList->append(secondPart->getPatient());
        secondPart = secondPart->getNextNode();
        secondPartSize--;
    }
}
}

```

*Figure 61 Merge sort method (Part 3)*

Figure 61 shows one part of the code snippet for the merge sort method under the Utility class.

```

else if (sortMode == 3) { // Compare patient time
    if (firstPart->getPatient()->getVisitHour() < secondPart->getPatient()->getVisitHour()) {
        sortedList->append(firstPart->getPatient());
        firstPart = firstPart->getNextNode();
        firstPartSize--;
    }
    else if (firstPart->getPatient()->getVisitHour() > secondPart->getPatient()->getVisitHour()) {
        sortedList->append(secondPart->getPatient());
        secondPart = secondPart->getNextNode();
        secondPartSize--;
    }
    else if (firstPart->getPatient()->getVisitMinute() < secondPart->getPatient()->getVisitMinute()) {
        sortedList->append(firstPart->getPatient());
        firstPart = firstPart->getNextNode();
        firstPartSize--;
    }
    else if (firstPart->getPatient()->getVisitMinute() > secondPart->getPatient()->getVisitMinute()) {
        sortedList->append(secondPart->getPatient());
        secondPart = secondPart->getNextNode();
        secondPartSize--;
    }
    else if (firstPart->getPatient()->getVisitSecond() <= secondPart->getPatient()->getVisitSecond()) {
        sortedList->append(firstPart->getPatient());
        firstPart = firstPart->getNextNode();
        firstPartSize--;
    }
    else {
        sortedList->append(secondPart->getPatient());
        secondPart = secondPart->getNextNode();
        secondPartSize--;
    }
}
}

```

*Figure 62 Merge sort method (Part 4)*

Figure 62 shows one part of the code snippet for the merge sort method under the Utility class.

```
        while (firstPartSize > 0) {
            sortedList->append(firstPart->getPatient());
            firstPart = firstPart->getNextNode();
            firstPartSize--;
        }
        while (secondPartSize > 0 && secondPart != NULL) {
            sortedList->append(secondPart->getPatient());
            secondPart = secondPart->getNextNode();
            secondPartSize--;
        }

        firstPart = secondPart;
    }
    if (totalMerges <= 1) {
        return sortedList;
    }
    segmentSize *= 2;
}
}
```

*Figure 63 Merge sort method (Part 5)*

Figure 63 shows the last part of the code snippet for the merge sort method under the Utility class. As the name suggests, the merge sort method applies the merge sort algorithm to sort linked lists. The merge sort method takes in a linked list and an integer which represents how to sort the linked lists as parameters (Figure 60) and return a sorted linked list if possible. Firstly, the merge sort method creates two pointers which will be used to point to the nodes that will be used to compare to determine which node precede first. By applying the merge sort concept, a linked list will become sorted by dividing and sorting the smaller parts of the linked list before merging all the smaller parts together. Therefore, the merge sort method starts by sorting each pair of nodes with a maximum size of 2. At the end of the sort, the linked list will be sorted in pairs with a maximum size of 2. If the number of sorts (merges) performed when sorting each pair of nodes with the maximum size of 2 exceeds 1, this means that the linked list is still not fully sorted yet. Therefore, the method continues by incrementing the size of the smaller parts by twice. At the next iteration, the merge sort method will be sorting each group of nodes with a maximum size of 4. The size continues to increase by twice until the number

of merges performed on the smaller parts of the linked list is equal to one. When the number of merges performed equals one, the linked list will be fully sorted. This is because when the total number of merges performed is equal to one, the linked list is broken into half and sorted, this means that every node will be sorted correctly, and it is the last iteration for any merge sort algorithm. Then, the fully sorted linked list will be returned.

```
std::string* Utility::getDoctors(int* numberOfDoctors)
{
    int rowsize = sizeof doctorCredentials / sizeof doctorCredentials[0];
    *numberOfDoctors = rowsize;
    std::string* doctornames = new std::string[rowsize];
    for (int i = 0; i < rowsize; i++)
    {
        doctornames[i] = doctorCredentials[i][0];
    }
    return doctornames;
}
```

*Figure 64 Get Doctors method*

The figure above displays the get doctors method in the utility class. The method accesses the array of doctor login credentials and extracts the username of every doctor's account. This is done using a for loop to traverse the credentials array and add the usernames into a separate string array. After the loop ends, the method returns the array of strings containing the names of all the doctors that are currently available in the clinic. In addition, the method also accepts a pointer to an integer variable, where the variables' value will be set as the total row size of the array so that it can be referenced later. An example of the utilisation of this method is in the add patient function in the nurse menu, where the nurse has to assign the patient to a doctor that is currently working in the clinic, which will be known after displaying the doctor names.

## Main Class

```

Patient* INIT_PATIENTQUEUE[5] = { new Patient("PID6", "Hui Lin", "Loo", "20", "female", "0132223194", "21,Jalan APU, Taman TPM", "Fever", "", "Ong", false),
    new Patient("PID7", "James", "Ethan", "30", "male", "0124428984", "21,Jalan APU, Taman TPM", "Migraine", "", "Ong", true),
    new Patient("PID8", "Chi En", "Chew", "21", "male", "0193873275", "21,Jalan APU, Taman TPM", "Malnutrition", "", "Wong", false),
    new Patient("PID9", "Bruh", "Kong", "25", "others", "0162985667", "21,Jalan APU, Taman TPM", "Infection", "", "Ong", true),
    new Patient("PID10", "Hua Iong", "Lee", "19", "male", "0193893758", "21,Jalan APU, Taman TPM", "Prostate disease", "", "Wong", false) };

Patient* INIT_HISTORYLIST[5] = { new Patient("PID1", "Hui Yin", "Loo", "18", "female", "0193875637", "21,Jalan APU, Taman TPM", "Fever", "", "Ong", false),
    new Patient("PID2", "Benson", "Junior", "17", "others", "0173846758", "21,Jalan APU, Taman TPM", "Migraine", "", "Ong", true),
    new Patient("PID3", "Chi En", "Ooi", "24", "female", "0183758275", "21,Jalan APU, Taman TPM", "Nauseous", "", "Wong", false),
    new Patient("PID4", "Kei Zhong", "Kong", "20", "male", "0162895784", "21,Jalan APU, Taman TPM", "Infection", "", "Ong", true),
    new Patient("PID5", "Hua Long", "Lee", "35", "male", "0163847182", "21,Jalan APU, Taman TPM", "Prostate disease", "Antibiotics", "Wong", false) };

int INIT_HISTORYTIME[5][3] = {
    {9,01,01},
    {9,30,59},
    {10,05,0},
    {13,00,00},
    {16,55,0},
};

int INIT_HISTORYDATE[5][3] = {
    {9,9,121},
    {10,9,121},
    {11,9,121},
    {13,9,121},
    {16,9,121},
};

```

*Figure 65 Arrays definitions*

Figure 65 shows four arrays defined in the main class. All four arrays contain patients' data that will be injected into the system once the system starts so that there exist some patients records in the system. The INIT\_PATIENTQUEUE contains patients that will be injected into the patient queue, while INIT\_HISTORYLIST contains patients that will be injected into the history list. The INIT\_HISTORYTIME is used to store the visit time for the 5 patients that will be injected into the history list because it is planned that the 5 patients in the history list will have a different visit time than each other and happens before the time when the system starts. Similarly, the INIT\_HISTORYDATE is used to store the visit day for the 5 patients that will be injected into the history list.



```

int main() {
    PatientQueue* p = new PatientQueue();
    LinkedList* h = new LinkedList();
    Doctor* doctor = new Doctor(p,h);
    Nurse* nurse = new Nurse(p,h);

    for (int i = 0; i < 5; i++)
    {
        p->insertPatient(INIT_PATIENTQUEUE[i]);
        INIT_HISTORYLIST[i]->setTime(INIT_HISTORYTIME[i][0], INIT_HISTORYTIME[i][1], INIT_HISTORYTIME[i][2]);
        INIT_HISTORYLIST[i]->setVisitDay(INIT_HISTORYDATE[i][0], INIT_HISTORYDATE[i][1], INIT_HISTORYDATE[i][2]);
        h->insertAtFront(INIT_HISTORYLIST[i]);
    }
    system("cls");
    std::string username;
    std::string password;
    char loginAsDoctor;
    int loginStatus = 1;
    while (true) {
        std::cout << "Login Page" << std::endl;
        std::cout << "Username : ";
        getline(std::cin, username);
        std::cout << "Password : ";
        password = Utility::getPassword();
        std::cout << "\nEnter 1 to login as doctor, otherwise enter any key to login as nurse :";
        std::cin >> loginAsDoctor;
        std::cin.ignore(256, '\n');
        loginStatus = Utility::login(username,password,loginAsDoctor,doctor,nurse);
        system("cls");
        if (loginStatus == -1) {
            std::cout << "Login failed. Invalid credentials provided. Please try again." << std::endl << std::endl;
            system("pause");
        }
        // Doctor
        else if (loginStatus == 0) {
            doctor->displayDoctorMenu();
        }
        else {
            //nurse
            nurse->displayNurseMenu();
        }
        system("cls");
    }
    return 0;
}

```

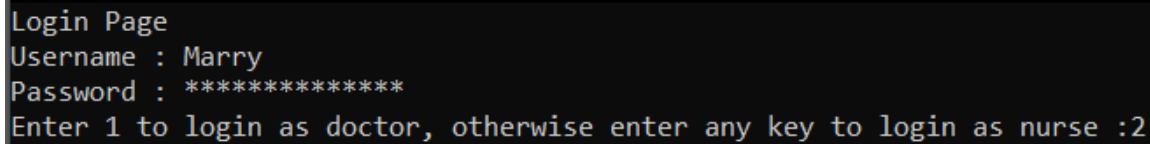
*Figure 66 Main method*

Figure 66 shows the code snippet for the main method. Firstly, when the program starts, it will initialize the patient queue and history list with the patients defined in the four arrays shown in Figure 65. Next, the method will display the login page in which the users have to enter their username and password. If the credentials are entered correctly, then the system will display

the nurse menu or doctor menu to the users according to the account logged in. Then, the users can choose the options to be performed by the system via the menu displayed.

## Screenshot of Inputs & Outputs

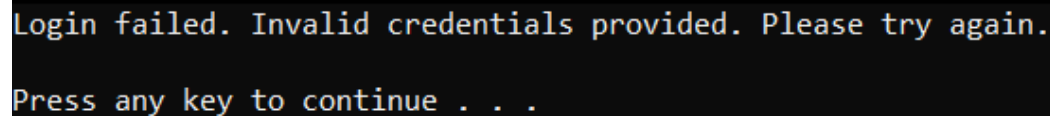
### Nurse Users



```
Login Page
Username : Marry
Password : *****
Enter 1 to login as doctor, otherwise enter any key to login as nurse :2
```

*Figure 67 Login Page (Nurse perspective)*

The figure above shows the login screen from the perspective of a nurse user. The program will prompt its user to enter their username and password as well as the login method. To log in as a nurse, the user will have to enter anything other than “1” for the 3<sup>rd</sup> input prompt. If credentials entered exist, the user will be logged in, and the nurse main menu will be displayed.



```
Login failed. Invalid credentials provided. Please try again.
Press any key to continue . . .
```

*Figure 68 Login Page invalid credentials error message*

The figure above shows the error message that will be displayed to the user if the login credentials entered are invalid.

```
=== Nurse Menu ===
1. View patient waiting list
2. Add patient to the waiting list
3. Call next patient for doctor visit
4. Search patient from waiting list
5. View patient waiting list sorted by time
6. View patient waiting list sorted by patient ID
7. Logout

Enter a number above: 1
Please enter [1] to view full waiting list, and [2] to view via page-by-page: 1
```

*Figure 69 Nurse view Main menu*

The figure above shows the main menu of the nurse menu while selecting the “view patient waiting list” option from the nurse main menu. When selected by entering ‘1’, the program will again prompt the user to enter the viewing mode for the list by either entering ‘1’ for a full view of the waiting list, or ‘2’ for a page-by-page view.

```
Here are all the patients in the waiting list:

Patient ID: PID7
Patient first Name: James
Patient last Name: Ethan
Patient Age: 30
Patient Gender: male
Patient Contact: 0124428984
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Migraine
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 01:01:43
-----
Patient ID: PID9
Patient first Name: Bruh
Patient last Name: Kong
Patient Age: 25
Patient Gender: others
Patient Contact: 0162985667
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Infection
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 01:01:43
-----
Patient ID: PID6
Patient first Name: Hui Lin
Patient last Name: Loe
Patient Age: 20
Patient Gender: female
Patient Contact: 0132223194
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Fever
Medicine Information:
Disabled : 0
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 01:01:43
-----
```

*Figure 70 Nurse display full waiting list output*

The figure above shows the output display from selecting the full-page viewing mode from the “view patient waiting list” option in the nurse menu.

```
Patient ID: PID7
Patient first Name: James
Patient last Name: Ethan
Patient Age: 30
Patient Gender: male
Patient Contact: 0124428984
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Migraine
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 01:01:43
Press :
[1] to view next page
[2] to exit viewing:
```

*Figure 71 Nurse viewing waiting list page-by-page output (1)*

The figure shows the page-by-page view for the patient waiting list from the nurses' menu. The output displays the details of the first patient at the start of the waiting queue and will prompt the user to enter a key to either view the next patient or exit the viewing.

```
Patient ID: PID9
Patient first Name: Bruh
Patient last Name: Kong
Patient Age: 25
Patient Gender: others
Patient Contact: 0162985667
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Infection
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 01:01:43
Press :
[0] to view previous page
[1] to view next page
[2] to exit viewing:
```

*Figure 72 Nurse viewing waiting list page-by-page output (2)*

The figure above shows the program output if the viewed patients' profile is not at the start of the waiting list in the page-by-page viewing mode.

```
Patient ID: PID7
Patient first Name: James
Patient last Name: Ethan
Patient Age: 30
Patient Gender: male
Patient Contact: 0124428984
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Migraine
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 01:01:43
Press :
[1] to view next page
[2] to exit viewing:
0

Sorry, but we do not support this input, please try again.

Press any key to continue . . .
```

*Figure 73 Nurse viewing patient list invalid input error message*

The figure displays the error code printed to the user if they entered an invalid input into the field.

```
=== Nurse Menu ===
1. View patient waiting list
2. Add patient to the waiting list
3. Call next patient for doctor visit
4. Search patient from waiting list
5. Logout

Enter a number above: 2_
```

*Figure 74 Nurse adding new patients to the waiting list*

The figure above shows the “Add patient to the waiting list” option from the nurse main menu. When selected by entering ‘2’, the program will again prompt the user to enter the add new patient mode.

```
-----Create patient menu-----  
  
Please enter the patient's first name: Wong  
Please enter the patient's last name: Jack Yih  
Please enter the patient's age: 20  
Please enter the patient's gender (Male/Female/Others): Male  
Please enter the patient's phone number: 0128244577  
Please enter the patient's address: Sabah,Malaysia  
Please enter the patient's sickness description: Flue  
Is the patient disabled? (Yes/No): No  
  
Available doctors:  
- Ong  
- Wong  
  
Please enter the patient's doctor name from above: Ong  
  
---Patient Details---  
Patient ID: PID11  
Patient first Name: Wong  
Patient last Name: Jack Yih  
Patient Age: 20  
Patient Gender: male  
Patient Contact: 0128244577  
Address : Sabah,Malaysia  
Responsible Doctor: Ong  
Sickness Description: Flue  
Medicine Information:  
Disabled : 0  
Visiting Day(dd/mm/yy) : 8/11/2021  
Visiting Time(hour / min / sec) : 20:11:17
```

*Figure 75 Nurse registering new patient*

Once the user has selected option 2, the user is required to enter all the relevant information that is necessary to successfully register a patient. Once all the required information has been entered, the system would give a summary under “Patient Details” tab of all the data that has been entered during patient registration, which also includes the visit day and visit time as well, which is auto assigned based on local system time.

```
Confirm adding patient into the waiting list? (Yes/No): yes  
  
Inserting patient into the waiting list...  
  
Patient added successfully
```

*Figure 76 Nurse confirming to add patient into the waiting queue*



Once the user has confirmed that all the information is correct, the user would confirm to add the patient into the waiting queue by typing “yes” into the system, this input is not case sensitive.

```
Confirm adding patient into the waiting list? (Yes/No): no
Process has been cancelled.
```

*Figure 77 Nurse rejects to add patient into the waiting queue*

If the user has made a mistake and is not satisfied with the patient information, the user could enter “No” during the confirmation stage before adding into the patient queue and could start over to re-enter the correct data and would show a “Process has been cancelled” message.

```
-----Create patient menu-----
Please enter the patient's first name: Wong
Please enter the patient's last name: Jack Yih
Please enter the patient's age: abc
Invalid input, please enter numeric values only.
```

*Figure 78 Nurse entering invalid data for age*

Several input validation methods have been put in place during the code explanation and justification section of the report, such as age as shown above.

```
Please enter the patient's age: 20
Please enter the patient's gender (Male/Female/Others): ABC
Invalid input, please try again with options given.
Please enter the patient's gender (Male/Female/Others): 123
```

*Figure 79 Nurse entering invalid data for gender*

The gender input section also has input validation set in place. It is important to note that the input for gender is not case-sensitive.

```
-----Create patient menu-----  
Please enter the patient's first name: Wong  
Please enter the patient's last name: Jack Yih  
Please enter the patient's age: 20  
Please enter the patient's gender (Male/Female/Others): Male  
Please enter the patient's phone number: 012-8244577  
  
Invalid input, please enter numeric values for phone numbers only.  
Please enter the patient's phone number: 0128244577abcd  
  
Invalid input, please enter numeric values for phone numbers only.  
Please enter the patient's phone number: 0128244577
```

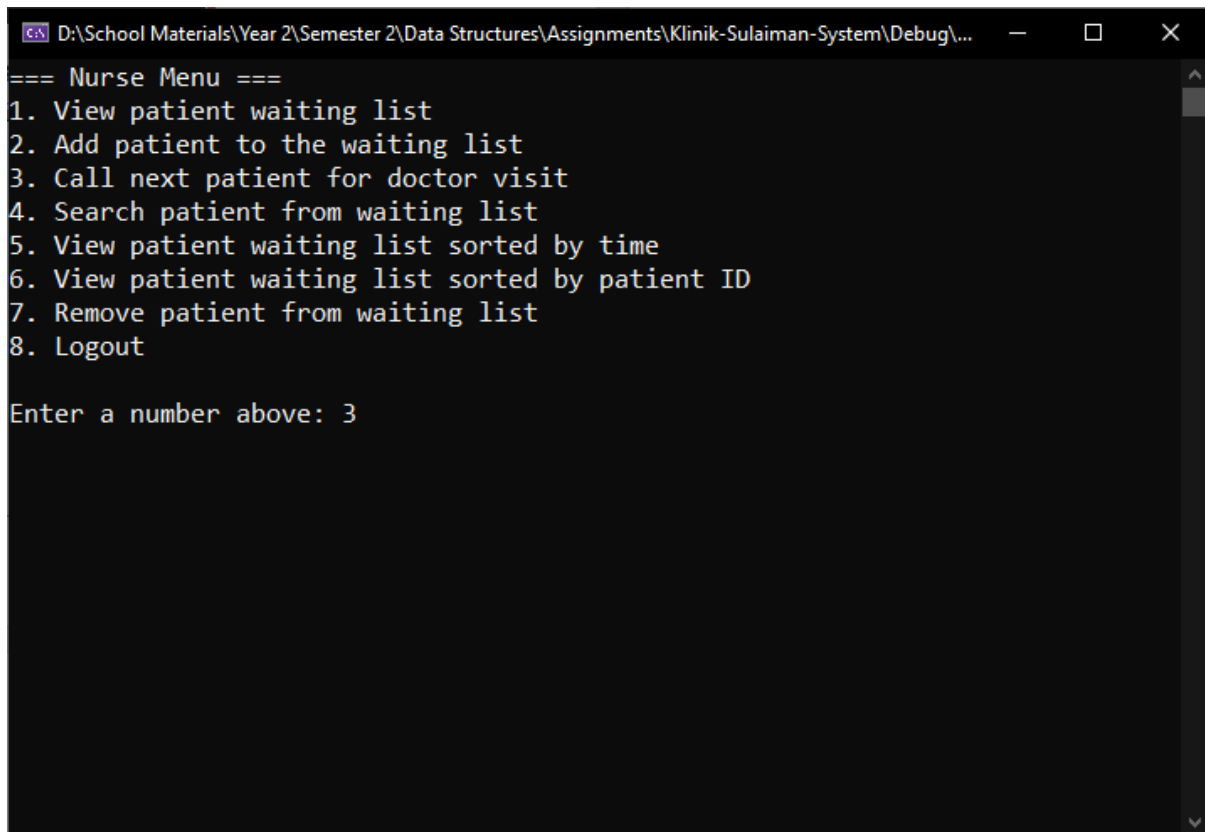
*Figure 80 Nurse entering invalid data for phone number*

The input validation for phone numbers would only accept numbers (Integer values) only, thus addition symbols such as “+” or “-” would not be accepted as well as alphabets as well.

```
Available doctors:  
- Ong  
- Wong  
  
Please enter the patient's doctor name from above: James  
  
Invalid input, please select a doctor from the list above!  
Please enter the patient's doctor name from above: ONG  
  
Invalid input, please select a doctor from the list above!  
Please enter the patient's doctor name from above:
```

*Figure 81 Nurse entering invalid doctor name*

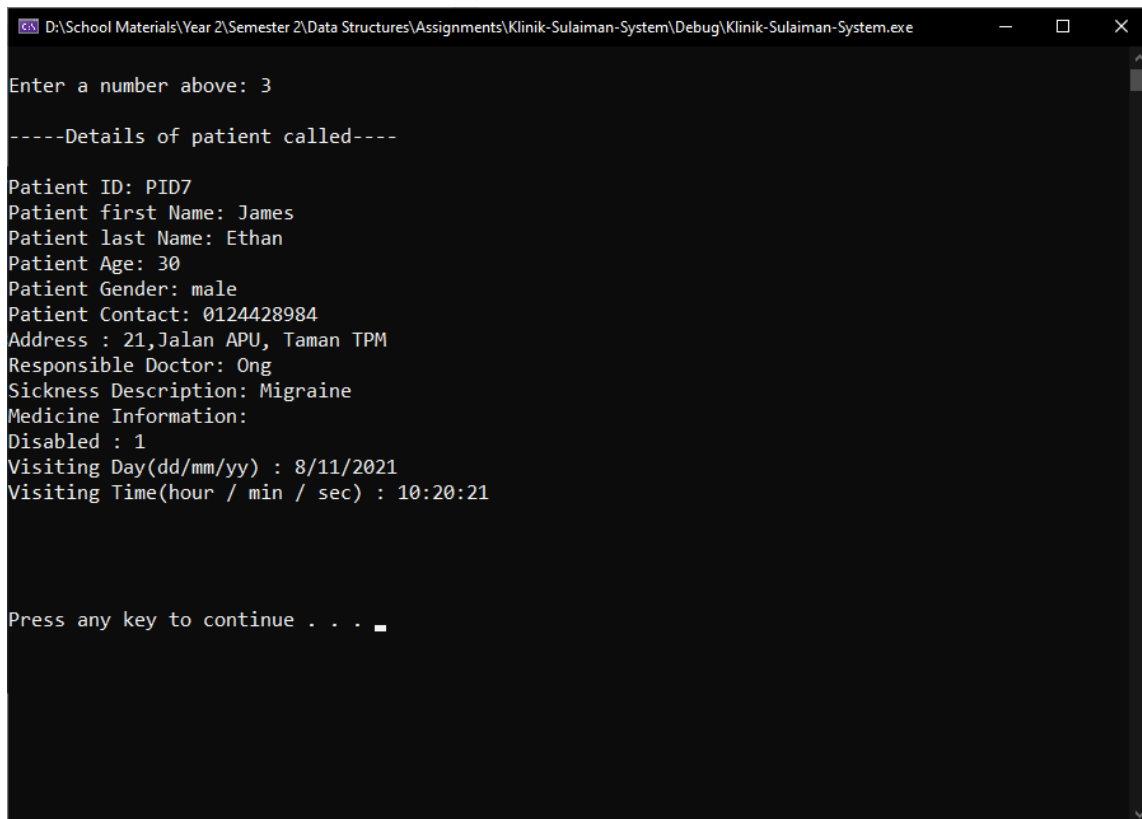
The input validation for doctor name would only accept the names as shown under the Available doctor’s tab, any variations or non-listed names would not be accepted, aside from that, the doctor’s name input is also case sensitive.



```
D:\School Materials\Year 2\Semester 2\Data Structures\Assignments\Klinik-Sulaiman-System\Debug\...  
=== Nurse Menu ===  
1. View patient waiting list  
2. Add patient to the waiting list  
3. Call next patient for doctor visit  
4. Search patient from waiting list  
5. View patient waiting list sorted by time  
6. View patient waiting list sorted by patient ID  
7. Remove patient from waiting list  
8. Logout  
  
Enter a number above: 3
```

*Figure 82 Nurse menu*

Figure 82 shows the nurse is using the system to call the next patient to visit the doctor by entering 3 to execute the call next patient command provided.

A screenshot of a Windows command prompt window. The title bar shows the file path: D:\School Materials\Year 2\Semester 2\Data Structures\Assignments\Klinik-Sulaiman-System\Debug\Klinik-Sulaiman-System.exe. The window contains the following text:

```
Enter a number above: 3  
-----Details of patient called-----  
Patient ID: PID7  
Patient first Name: James  
Patient last Name: Ethan  
Patient Age: 30  
Patient Gender: male  
Patient Contact: 0124428984  
Address : 21,Jalan APU, Taman TPM  
Responsible Doctor: Ong  
Sickness Description: Migraine  
Medicine Information:  
Disabled : 1  
Visiting Day(dd/mm/yy) : 8/11/2021  
Visiting Time(hour / min / sec) : 10:20:21  
  
Press any key to continue . . . █
```

*Figure 83 Patient details screen*

Figure 83 shows the patient details that is called for doctor visit from the patient queue. The patient with the details displayed in Figure 83 will be removed from the patient queue and inserted into the patient visit history list.

```
=== Nurse Menu ===  
1. View patient waiting list  
2. Add patient to the waiting list  
3. Call next patient for doctor visit  
4. Search patient from waiting list  
5. View patient waiting list sorted by time  
6. View patient waiting list sorted by patient ID  
7. Remove patient from waiting list  
8. Logout  
  
Enter a number above: 4
```

*Figure 84 Nurse selecting 4<sup>th</sup> option from main menu*

The figure above shows the user selecting the 4<sup>th</sup> option to search for patients within the waiting list.

```
=== Nurse Searching Menu ===  
1. Search by Patient ID  
2. Search by Patient First Name  
3. Go Back  
  
Please select an option to search for the patients' profile: _
```

*Figure 85 Nurse view of the searching sub-menu*

The figure above shows the system asking the user to input which search modes to use.

```
Please select an option to search for the patients' profile: 1

Please enter the patient details to be searched with: PID11

Patient ID: PID11
Patient first Name: Wong
Patient last Name: Jack Yih
Patient Age: 20
Patient Gender: male
Patient Contact: 0128244577
Address : Sabah,Malaysia
Responsible Doctor: Ong
Sickness Description: Flue
Medicine Information:
Disabled : 0
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 10:27:49
-----
```

*Figure 86 Nurse view after entering search mode and search reference (Part 1)*

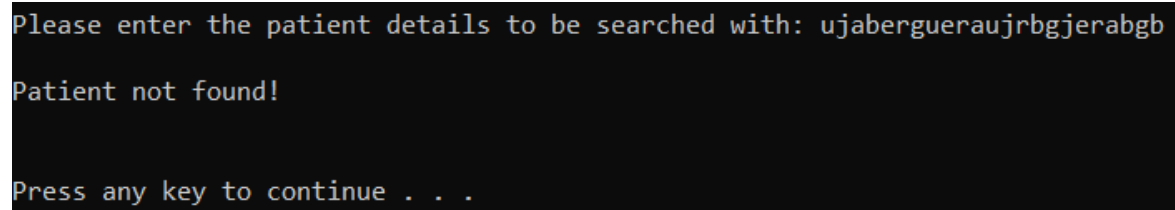
```
Please select an option to search for the patients' profile: 2

Please enter the patient details to be searched with: Wong

Patient ID: PID11
Patient first Name: Wong
Patient last Name: Jack Yih
Patient Age: 20
Patient Gender: male
Patient Contact: 0128244577
Address : Sabah,Malaysia
Responsible Doctor: Ong
Sickness Description: Flue
Medicine Information:
Disabled : 0
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 10:27:49
-----
```

*Figure 87 Nurse view after entering search mode and search reference (Part 2)*

The figure above shows the search result outputs based on the user input for search reference and the search mode.

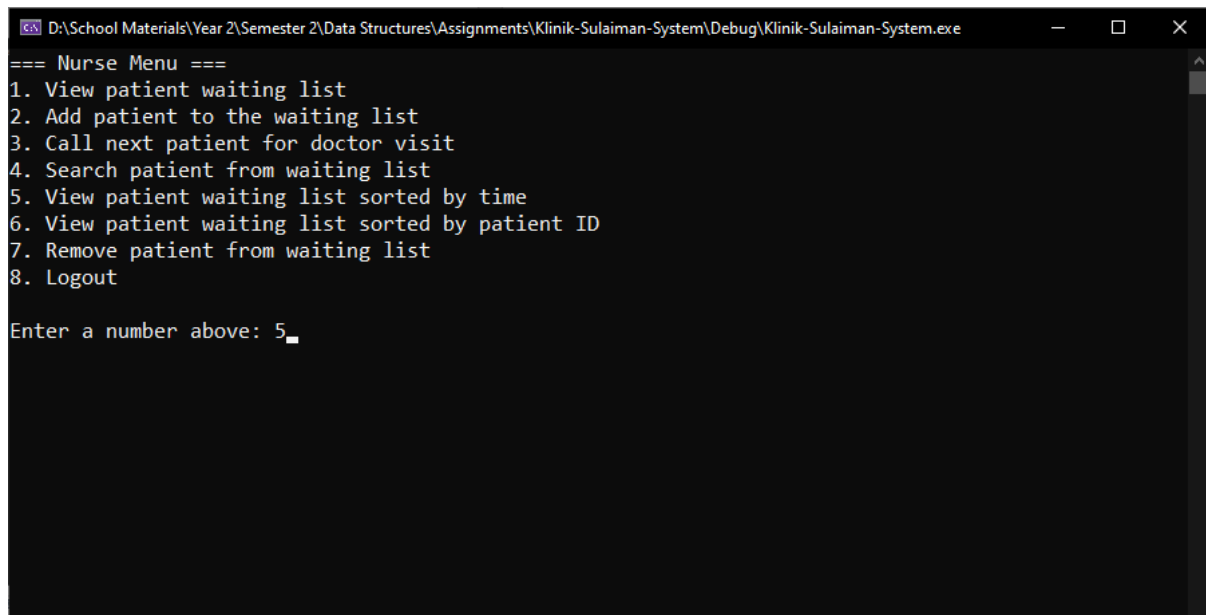
A terminal window with a black background and white text. The text reads: "Please enter the patient details to be searched with: ujabergueraujrbgjerabgb", "Patient not found!", and "Press any key to continue . . .".

```
Please enter the patient details to be searched with: ujabergueraujrbgjerabgb
Patient not found!
Press any key to continue . . .
```

*Figure 88 Nurse view when no matching records have been found*

The figure above shows the output in the event that there is no matching data when comparing the user input against the data within the linked list.



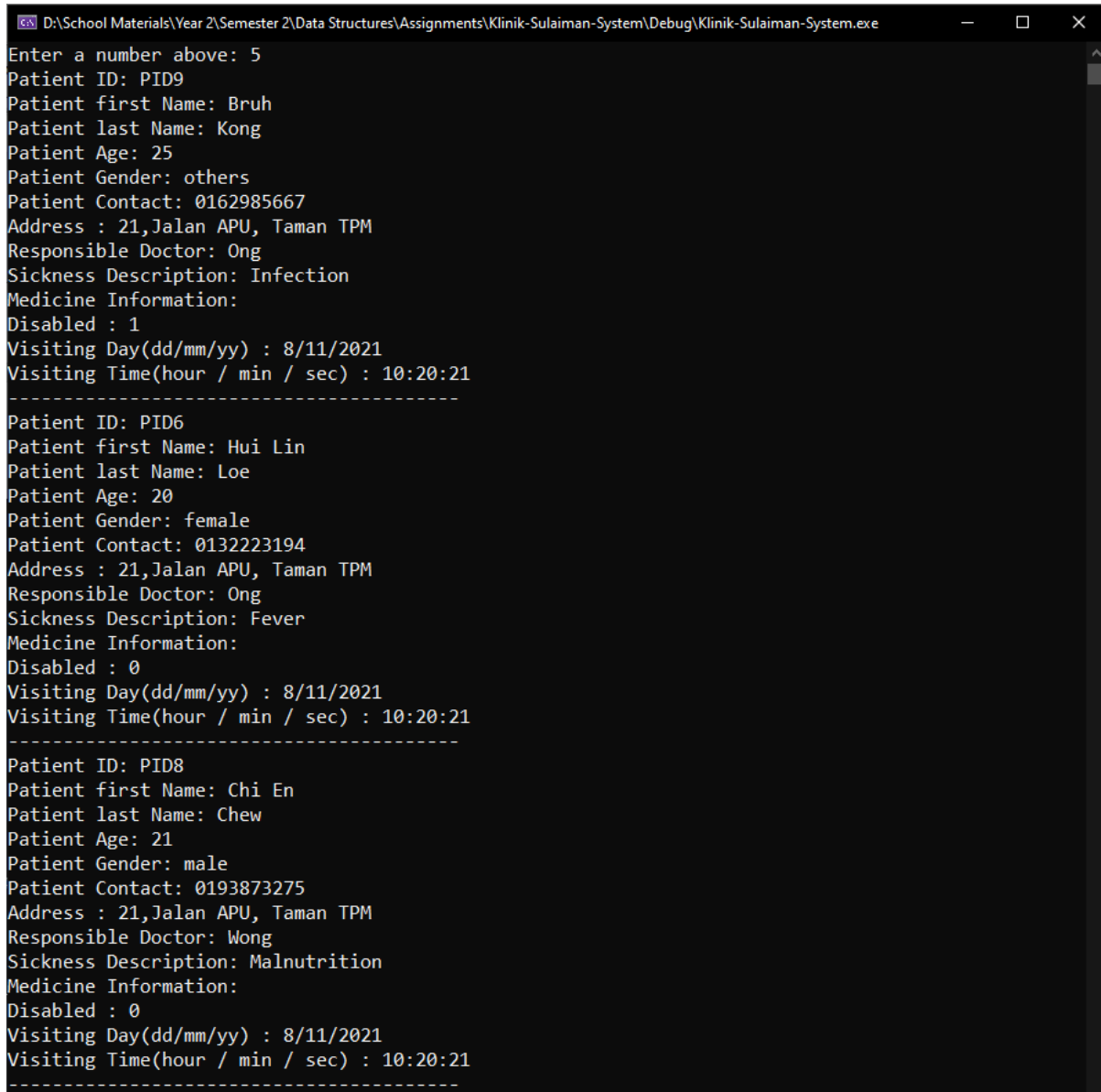


```
D:\School Materials\Year 2\Semester 2\Data Structures\Assignments\Klinik-Sulaiman-System\Debug\Klinik-Sulaiman-System.exe
=== Nurse Menu ===
1. View patient waiting list
2. Add patient to the waiting list
3. Call next patient for doctor visit
4. Search patient from waiting list
5. View patient waiting list sorted by time
6. View patient waiting list sorted by patient ID
7. Remove patient from waiting list
8. Logout

Enter a number above: 5_
```

*Figure 89 Nurse menu*

Figure 89 shows the nurse can view the patient waiting list sorted by time by entering 5 in the nurse menu.



```
D:\School Materials\Year 2\Semester 2\Data Structures\Assignments\Klinik-Sulaiman-System\Debug\Klinik-Sulaiman-System.exe
Enter a number above: 5
Patient ID: PID9
Patient first Name: Bruh
Patient last Name: Kong
Patient Age: 25
Patient Gender: others
Patient Contact: 0162985667
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Infection
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 10:20:21
-----
Patient ID: PID6
Patient first Name: Hui Lin
Patient last Name: Loe
Patient Age: 20
Patient Gender: female
Patient Contact: 0132223194
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Fever
Medicine Information:
Disabled : 0
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 10:20:21
-----
Patient ID: PID8
Patient first Name: Chi En
Patient last Name: Chew
Patient Age: 21
Patient Gender: male
Patient Contact: 0193873275
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Wong
Sickness Description: Malnutrition
Medicine Information:
Disabled : 0
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 10:20:21
-----
```

*Figure 90 Output screen for viewing patient waiting list sorted by time (Part 1)*

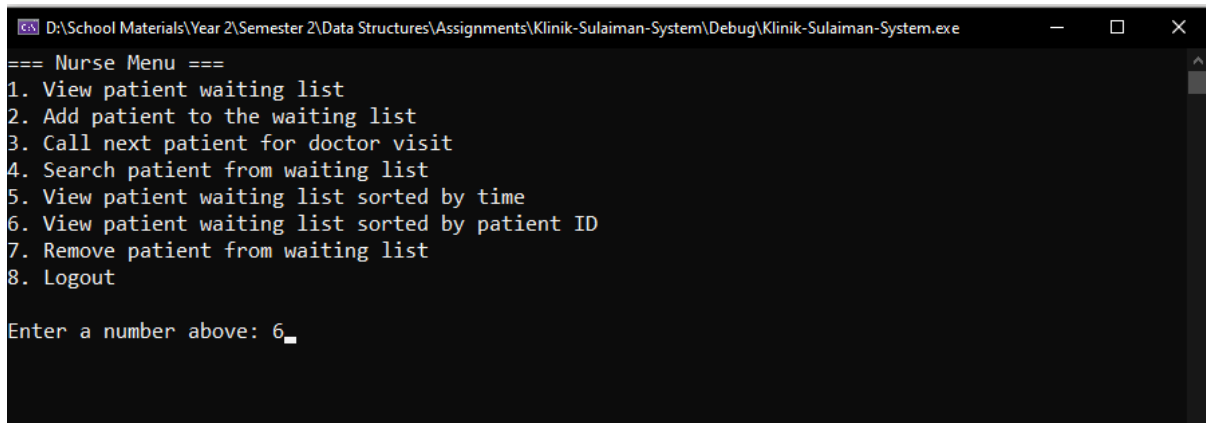
Figure 90 shows the part of the output screen that displays the patient waiting list with patients sorted by time.

```
-----
Patient ID: PID10
Patient first Name: Hua Iong
Patient last Name: Lee
Patient Age: 19
Patient Gender: male
Patient Contact: 0193893758
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Wong
Sickness Description: Prostate disease
Medicine Information:
Disabled : 0
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 10:20:21
-----
Patient ID: PID11
Patient first Name: Jane
Patient last Name: Foster
Patient Age: 34
Patient Gender: female
Patient Contact: 0129999999
Address : 23,Jalan Permaisuri, Taman Tong
Responsible Doctor: Ong
Sickness Description: Mild fever
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 10:33:53
-----

Press any key to continue . . . █
```

*Figure 91 Output screen for viewing patient waiting list sorted by time (Part 2)*

Figure 91 shows the last part of the output screen that displays the patient waiting list with patients sorted by time. It can be observed that the last patient Jane Foster has the latest visiting time with a disabled status. Therefore, it proves that the sorting works, otherwise the system will place disabled patients before non-disabled patients.



```
D:\School Materials\Year 2\Semester 2\Data Structures\Assignments\Klinik-Sulaiman-System\Debug\Klinik-Sulaiman-System.exe
=== Nurse Menu ===
1. View patient waiting list
2. Add patient to the waiting list
3. Call next patient for doctor visit
4. Search patient from waiting list
5. View patient waiting list sorted by time
6. View patient waiting list sorted by patient ID
7. Remove patient from waiting list
8. Logout

Enter a number above: 6_
```

*Figure 92 Nurse menu*

Figure 92 shows the nurse can view the patient waiting list sorted by patient ID by entering command 6 at the nurse menu.

```
Enter a number above: 6
Patient ID: PID6
Patient first Name: Hui Lin
Patient last Name: Loe
Patient Age: 20
Patient Gender: female
Patient Contact: 0132223194
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Fever
Medicine Information:
Disabled : 0
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 10:20:21
-----
Patient ID: PID8
Patient first Name: Chi En
Patient last Name: Chew
Patient Age: 21
Patient Gender: male
Patient Contact: 0193873275
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Wong
Sickness Description: Malnutrition
Medicine Information:
Disabled : 0
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 10:20:21
-----
Patient ID: PID9
Patient first Name: Bruh
Patient last Name: Kong
Patient Age: 25
Patient Gender: others
Patient Contact: 0162985667
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Infection
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 10:20:21
-----
```

*Figure 93 Output screen for patient waiting list sorted by patient ID (Part 1)*

Figure 93 shows one part of the output screen that displays the patient waiting list with patients sorted by patient ID.

```
Patient ID: PID10
Patient first Name: Hua Iong
Patient last Name: Lee
Patient Age: 19
Patient Gender: male
Patient Contact: 0193893758
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Wong
Sickness Description: Prostate disease
Medicine Information:
Disabled : 0
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 10:20:21
-----
Patient ID: PID11
Patient first Name: Jane
Patient last Name: Foster
Patient Age: 34
Patient Gender: female
Patient Contact: 0129999999
Address : 23,Jalan Permaisuri, Taman Tong
Responsible Doctor: Ong
Sickness Description: Mild fever
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 10:33:53
-----

Press any key to continue . . . ■
```

*Figure 94 Output screen for patient waiting list sorted by patient ID (Part 2)*

Figure 94 shows the last part of the output screen that displays the patient waiting list with patients sorted by patient ID. It can be observed that the patients are indeed sorted by patient ID from Figure 93 and Figure 94.

```
=== Nurse Menu ===  
1. View patient waiting list  
2. Add patient to the waiting list  
3. Call next patient for doctor visit  
4. Search patient from waiting list  
5. View patient waiting list sorted by time  
6. View patient waiting list sorted by patient ID  
7. Remove patient from waiting list  
8. Logout  
  
Enter a number above: 7  
  
Enter ID of patient to be deleted :
```

*Figure 95 Remove patient option in nurse menu*

The figure above shows the remove patient option from the nurse main menu. Once selected, the program will prompt the user to enter the ID of the patient which they intend to remove from the waiting list.

```
=== Nurse Menu ===  
1. View patient waiting list  
2. Add patient to the waiting list  
3. Call next patient for doctor visit  
4. Search patient from waiting list  
5. View patient waiting list sorted by time  
6. View patient waiting list sorted by patient ID  
7. Remove patient from waiting list  
8. Logout  
  
Enter a number above: 7  
  
Enter ID of patient to be deleted : PID6  
  
Patient with ID 'PID6' successfully deleted.  
  
Press any key to continue . . .
```

*Figure 96 Successful patient deletion output*

The figure above displays the system output after the nurse user enters an ID of an existing patient on the waiting list. The program will notify the user that the patient profile has been successfully removed and the method ends.

```
=== Nurse Menu ===
1. View patient waiting list
2. Add patient to the waiting list
3. Call next patient for doctor visit
4. Search patient from waiting list
5. View patient waiting list sorted by time
6. View patient waiting list sorted by patient ID
7. Remove patient from waiting list
8. Logout

Enter a number above: 7

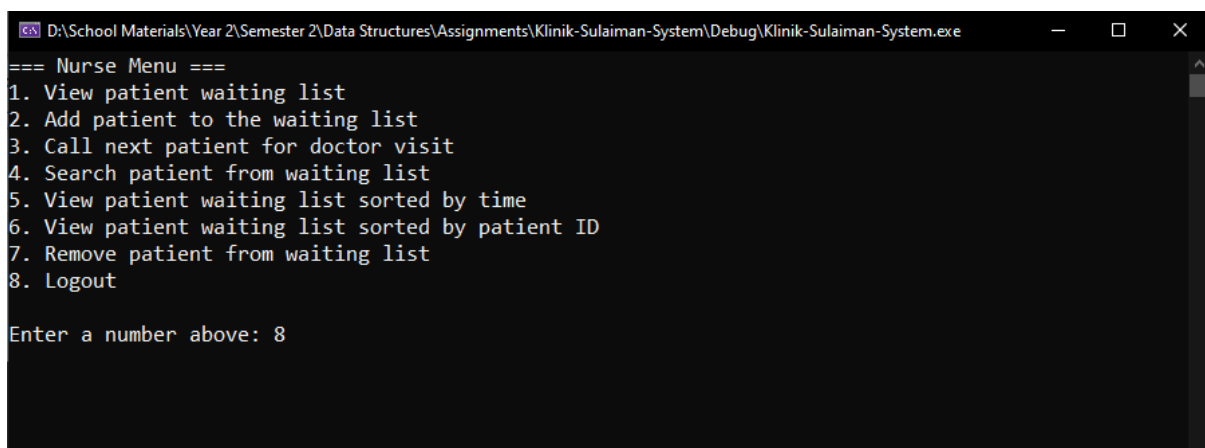
Enter ID of patient to be deleted : PID1

Invalid Patient ID or Patient with this ID does not exist...

Press any key to continue . . .
```

*Figure 97 Patient ID not found error message*

The figure above shows the error message displayed to the user when they passed in a patient ID that does not exist from any patient currently on the waiting list.



```
D:\School Materials\Year 2\Semester 2\Data Structures\Assignments\Klinik-Sulaiman-System\Debug\Klinik-Sulaiman-System.exe
=== Nurse Menu ===
1. View patient waiting list
2. Add patient to the waiting list
3. Call next patient for doctor visit
4. Search patient from waiting list
5. View patient waiting list sorted by time
6. View patient waiting list sorted by patient ID
7. Remove patient from waiting list
8. Logout

Enter a number above: 8
```

*Figure 98 Nurse menu*

Figure 98 shows the nurse can log out from the system by entering command 8. The system will then proceed to change the screen back to the login page as shown in Figure 99.

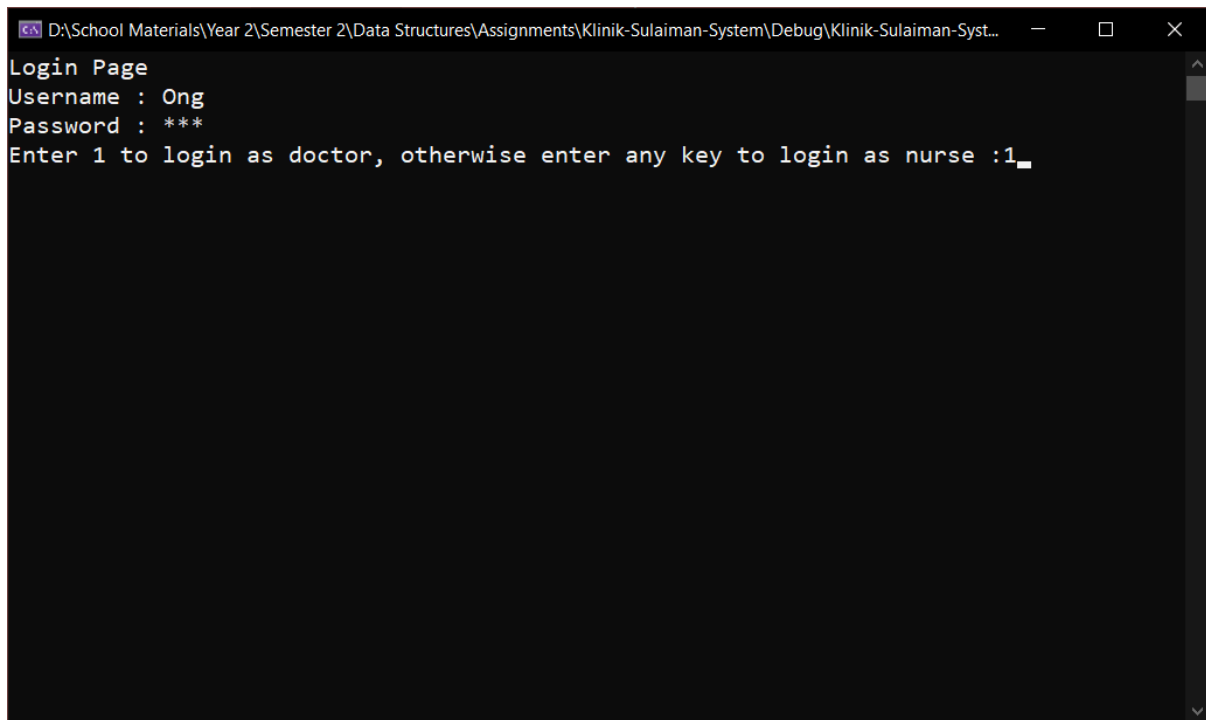


## Doctor User



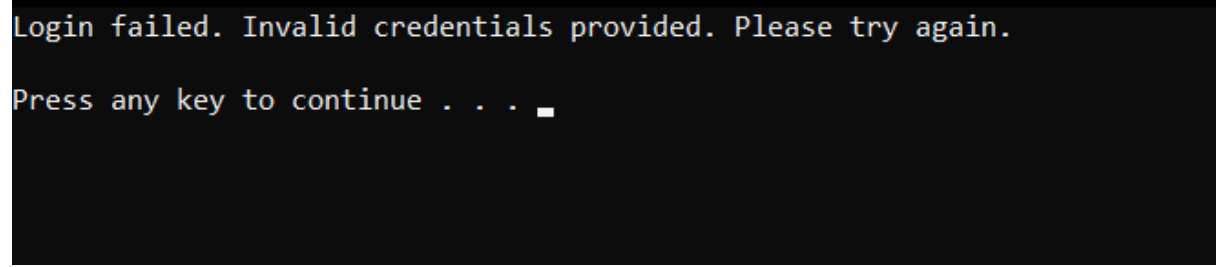
*Figure 99 Login Page*

Figure 99 shows the login page prompting the doctor to enter his or her username.



*Figure 100 Login Page with details entered*

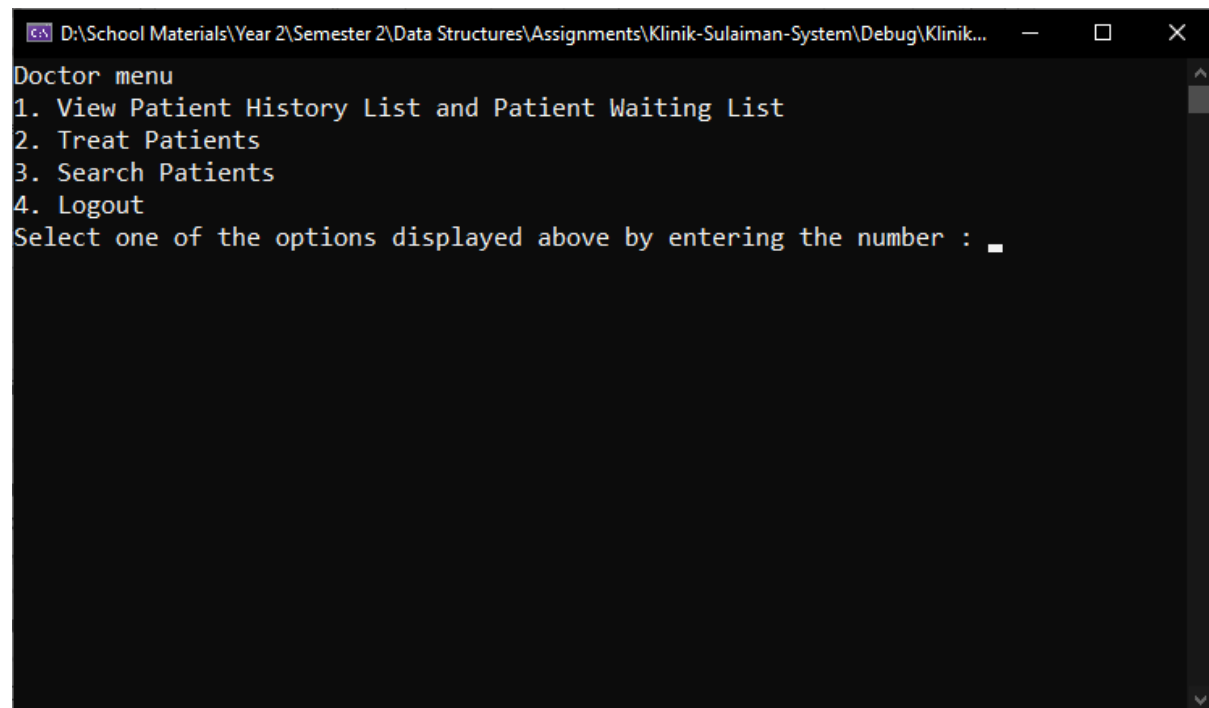
Figure 100 shows the doctor has entered his or her username and password. Then, the system will prompt the doctor to select whether to log in as a doctor or a nurse with the credentials provided.



```
Login failed. Invalid credentials provided. Please try again.  
Press any key to continue . . . █
```

*Figure 101 Login failed page*

Figure 101 shows the login failed page when the system cannot verify the credentials provided by the users. The users can try again by pressing any key, which the system will redirect the users back to the login page.



```
Doctor menu  
1. View Patient History List and Patient Waiting List  
2. Treat Patients  
3. Search Patients  
4. Logout  
Select one of the options displayed above by entering the number : █
```

*Figure 102 Doctor's menu*

Figure 102 shows the doctor's menu. Once the login process is successful, the doctor menu is displayed to the user if the user chose to login as a doctor.

```
Doctor menu
1. View Patient History List and Patient Waiting List
2. Treat Patients
3. Search Patients
4. Logout
Select one of the options displayed above by entering the number : 1_
```

*Figure 103 Doctor menu*

Figure 103 shows the doctor entering the viewing menu by selecting option 1.

```
=== Doctor Viewing Menu ===
1. View entire patient waiting list
2. View patient waiting list in page by page mode
3. View entire patient visit history list (Descending order by visit time)
4. View patient visit history list in page by page mode
5. View sorted history list based on patient first name
6. View sorted waiting list based on patient first name
7. View sorted history list based on sickness description
8. View sorted waiting list based on sickness description
9. Go Back to previous menu
Enter a number above:
```

*Figure 104 Doctor view patient menu*

The figure above shows the doctors' viewing menu where the doctor can choose from 8 viewing options for searching either the patient waiting list or the patient visit history. The options include viewing the lists of patients in a full list or page-by-page view and viewing the patients by their original sequence or sorted depending on their first name or sickness description.

```
=== Doctor Viewing Menu ===
1. View entire patient waiting list
2. View patient waiting list in page by page mode
3. View entire patient visit history list (Descending order by visit time)
4. View patient visit history list in page by page mode
5. View sorted history list based on patient first name
6. View sorted waiting list based on patient first name
7. View sorted history list based on sickness description
8. View sorted waiting list based on sickness description
9. Go Back to previous menu
Enter a number above: a

WARNING: The input you have entered is not supported by the system, please select from the menu
Press any key to continue . . .
```

*Figure 105 Doctor viewing menu input error message*

The figure above shows the error message displayed when the doctor user enters an invalid input in the viewing menu options.

```
=== Doctor Viewing Menu ===
1. View entire patient waiting list
2. View patient waiting list in page by page mode
3. View entire patient visit history list (Descending order by visit time)
4. View patient visit history list in page by page mode
5. View sorted history list based on patient first name
6. View sorted waiting list based on patient first name
7. View sorted history list based on sickness description
8. View sorted waiting list based on sickness description
9. Go Back to previous menu

Enter a number above: 1

Patient ID: PID7
Patient first Name: James
Patient last Name: Ethan
Patient Age: 30
Patient Gender: male
Patient Contact: 0124428984
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Migraine
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 01:01:43
-----
Patient ID: PID9
Patient first Name: Bruh
Patient last Name: Kong
Patient Age: 25
Patient Gender: others
Patient Contact: 0162985667
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Infection
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 01:01:43
-----
Patient ID: PID6
Patient first Name: Hui Lin
Patient last Name: Loe
Patient Age: 20
Patient Gender: female
```

*Figure 106 View full patient waiting list output*

The figure above shows the program output when the doctor user chooses to view the entire current patient waiting list, where all the details of the patients are displayed.

```
=== Doctor Viewing Menu ===
1. View entire patient waiting list
2. View patient waiting list in page by page mode
3. View entire patient visit history list (Descending order by visit time)
4. View patient visit history list in page by page mode
5. View sorted history list based on patient first name
6. View sorted waiting list based on patient first name
7. View sorted history list based on sickness description
8. View sorted waiting list based on sickness description
9. Go Back to previous menu

Enter a number above: 2_
```

*Figure 107 Doctor choosing the 2<sup>nd</sup> option*

Figure 107 shows doctor choosing the option 2 in the doctor viewing menu.

```
Patient ID: PID7
Patient first Name: James
Patient last Name: Ethan
Patient Age: 30
Patient Gender: male
Patient Contact: 0124428984
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Migraine
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 01:01:43
Press :
[1] to view next page
[2] to exit viewing:
```

*Figure 108 Doctor's patient waiting list page-by-page view (1)*

The figure shows the page-by-page view for the patient waiting list from the doctor search menu. The output displays the details of the first patient at the start of the waiting queue and will prompt the user to enter a key to either view the next patient or exit the viewing.

```
Patient ID: PID9
Patient first Name: Bruh
Patient last Name: Kong
Patient Age: 25
Patient Gender: others
Patient Contact: 0162985667
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Infection
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 01:01:43
Press :
[0] to view previous page
[1] to view next page
[2] to exit viewing:
```

*Figure 109 Doctor's patient waiting list page-by-page view (2)*

The figure above shows the program output if the viewing a patients' profile that is not at the start of the waiting list via the page-by-page viewing mode, where they will be given an additional option to enter '0' to view the previous patient.

```
=== Doctor Viewing Menu ===
1. View entire patient waiting list
2. View patient waiting list in page by page mode
3. View entire patient visit history list (Descending order by visit time)
4. View patient visit history list in page by page mode
5. View sorted history list based on patient first name
6. View sorted waiting list based on patient first name
7. View sorted history list based on sickness description
8. View sorted waiting list based on sickness description
9. Go Back to previous menu

Enter a number above: 6

Patient ID: PID9
Patient first Name: Bruh
Patient last Name: Kong
Patient Age: 25
Patient Gender: others
Patient Contact: 0162985667
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Infection
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 01:01:43
-----
Patient ID: PID8
Patient first Name: Chi En
Patient last Name: Chew
Patient Age: 21
Patient Gender: male
Patient Contact: 0193873275
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Wong
Sickness Description: Malnutrition
Medicine Information:
Disabled : 0
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 01:01:43
-----
Patient ID: PID10
Patient first Name: Hua Iong
Patient last Name: Lee
Patient Age: 19
Patient Gender: male
```

*Figure 110 View full patient waiting list sorted by patient first name output*

The figure above shows the system output when the user chooses to view the history waiting list based on the patients' first name by entering '6'. When comparing the output from selecting option 1 from above, it is apparent that this option provides a different sequence of output as the first names of the patients are sorted in alphabetical order.



```
=== Doctor Viewing Menu ===
1. View entire patient waiting list
2. View patient waiting list in page by page mode
3. View entire patient visit history list (Descending order by visit time)
4. View patient visit history list in page by page mode
5. View sorted history list based on patient first name
6. View sorted waiting list based on patient first name
7. View sorted history list based on sickness description
8. View sorted waiting list based on sickness description
9. Go Back to previous menu

Enter a number above: 8

Patient ID: PID6
Patient first Name: Hui Lin
Patient last Name: Loe
Patient Age: 20
Patient Gender: female
Patient Contact: 0132223194
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Fever
Medicine Information:
Disabled : 0
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 01:01:43
-----
Patient ID: PID9
Patient first Name: Bruh
Patient last Name: Kong
Patient Age: 25
Patient Gender: others
Patient Contact: 0162985667
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Infection
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 01:01:43
-----
Patient ID: PID8
Patient first Name: Chi En
Patient last Name: Chew
Patient Age: 21
Patient Gender: male
Patient Contact: 0193873275
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Wong
Sickness Description: Malnutrition
Medicine Information:
Disabled : 0
```

*Figure 111 View full patient waiting list sorted by patient sickness description output*

The figure above shows the system output when the user chooses to view the history waiting list based on the patients' sickness description by entering '8'. It can be seen that the output produced by this mode is different from the previous methods as the sequence of displaying patient profiles here is based on the alphabetical order of the patients' sickness description.

```
=== Doctor Viewing Menu ===
1. View entire patient waiting list
2. View patient waiting list in page by page mode
3. View entire patient visit history list (Descending order by visit time)
4. View patient visit history list in page by page mode
5. View sorted history list based on patient first name
6. View sorted waiting list based on patient first name
7. View sorted history list based on sickness description
8. View sorted waiting list based on sickness description
9. Go Back to previous menu

Enter a number above: 3

Patient ID: PID5
Patient first Name: Hua Long
Patient last Name: Lee
Patient Age: 35
Patient Gender: male
Patient Contact: 0163847182
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Wong
Sickness Description: Prostate disease
Medicine Information: Antibiotics
Disabled : 0
Visiting Day(dd/mm/yy) : 16/10/2021
Visiting Time(hour / min / sec) : 16:55:00
-----
Patient ID: PID4
Patient first Name: Kei Zhong
Patient last Name: Kong
Patient Age: 20
Patient Gender: male
Patient Contact: 0162895784
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Infection
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 13/10/2021
Visiting Time(hour / min / sec) : 13:00:00
-----
Patient ID: PID3
Patient first Name: Chi En
Patient last Name: Ooi
Patient Age: 24
Patient Gender: female
Patient Contact: 0183758275
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Wong
Sickness Description: Nauseous
Medicine Information:
Disabled : 0
```

*Figure 112 Doctor view after selecting option 3 viewing method*

The figure above shows the entire history list in descending order based on patient visit time once the doctor has selected the 3<sup>rd</sup> viewing option.

```
=== Doctor Viewing Menu ===
1. View entire patient waiting list
2. View patient waiting list in page by page mode
3. View entire patient visit history list (Descending order by visit time)
4. View patient visit history list in page by page mode
5. View sorted history list based on patient first name
6. View sorted waiting list based on patient first name
7. View sorted history list based on sickness description
8. View sorted waiting list based on sickness description
9. Go Back to previous menu

Enter a number above: 4_
```

*Figure 113 Doctor view on selecting option 4 in the viewing sub-menu*

The figure above shows the doctor selecting the 4<sup>th</sup> viewing option, which would bring the user directly to view the history list in page-by-page mode as shown in the figure below.

```
Patient ID: PID5
Patient first Name: Hua Long
Patient last Name: Lee
Patient Age: 35
Patient Gender: male
Patient Contact: 0163847182
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Wong
Sickness Description: Prostate disease
Medicine Information: Antibiotics
Disabled : 0
Visiting Day(dd/mm/yy) : 16/10/2021
Visiting Time(hour / min / sec) : 16:55:00
Press :
[1] to view next page
[2] to exit viewing:
```

*Figure 114 Doctor view when traversing from the head node*

The figure above shows the patient data within the history list in page-by-page mode, the image above shows only 2 options to navigate the program, this depends on the position of the node within the linked list, in this case, since it is printing from the head node, only moving to the next page and exit viewing would be shown.

```
Patient ID: PID4
Patient first Name: Kei Zhong
Patient last Name: Kong
Patient Age: 20
Patient Gender: male
Patient Contact: 0162895784
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Infection
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 13/10/2021
Visiting Time(hour / min / sec) : 13:00:00
Press :
[0] to view previous page
[1] to view next page
[2] to exit viewing:
1
```

*Figure 115 Doctor view when traversing in the middle of the linked list*

The figure above shows the patient data within the history list in the page-by-page mode which is similar to the previous figure, however now the user has access to choose all 3 options since the node position that it is printing from is between the head and tail node.

```
Patient ID: PID1
Patient first Name: Hui Yin
Patient last Name: Loe
Patient Age: 18
Patient Gender: female
Patient Contact: 0193875637
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Fever
Medicine Information:
Disabled : 0
Visiting Day(dd/mm/yy) : 9/10/2021
Visiting Time(hour / min / sec) : 09:01:01
Press :
[0] to view previous page
[2] to exit viewing:
```

*Figure 116 Doctor view when traversing to the tail node of the linked list*

The figure above shows the patient data within the history list in page-by-page mode, the following options will only be shown when the user has reached the end of the history list.

```
Press :
[0] to view previous page
[2] to exit viewing:
1

Sorry, but we do not support this input, please try again.

Press any key to continue . . . ■
```

*Figure 117 Doctor view when entering options not listed in the option menu*

The figure above shows the error message when the user has entered an option that is not displayed in the options menu, the following error message will also be shown when the user types in other inputs that are not integer values.

```
=== Doctor Viewing Menu ===
1. View entire patient waiting list
2. View patient waiting list in page by page mode
3. View entire patient visit history list (Descending order by visit time)
4. View patient visit history list in page by page mode
5. View sorted history list based on patient first name
6. View sorted waiting list based on patient first name
7. View sorted history list based on sickness description
8. View sorted waiting list based on sickness description
9. Go Back to previous menu

Enter a number above: 5

Patient ID: PID2
Patient first Name: Benson
Patient last Name: Junior
Patient Age: 17
Patient Gender: others
Patient Contact: 0173846758
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Migraine
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 10/10/2021
Visiting Time(hour / min / sec) : 09:30:59
-----
Patient ID: PID3
Patient first Name: Chi En
Patient last Name: Ooi
Patient Age: 24
Patient Gender: female
Patient Contact: 0183758275
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Wong
Sickness Description: Nauseous
Medicine Information:
Disabled : 0
Visiting Day(dd/mm/yy) : 11/10/2021
Visiting Time(hour / min / sec) : 10:05:00
-----
Patient ID: PID5
Patient first Name: Hua Long
Patient last Name: Lee
Patient Age: 35
Patient Gender: male
Patient Contact: 0163847182
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Wong
Sickness Description: Prostate disease
Medicine Information: Antibiotics
Disabled : 0
```

*Figure 118 Doctor view on selecting option 5 in the viewing sub-menu*

The figure above shows the output of a sorted history list based on the patient's first name. This output will only be shown when the user has selected option 5. The list that is displayed is sorted in ascending order based on the first alphabet of the patient's first name.

```
=== Doctor Viewing Menu ===
1. View entire patient waiting list
2. View patient waiting list in page by page mode
3. View entire patient visit history list (Descending order by visit time)
4. View patient visit history list in page by page mode
5. View sorted history list based on patient first name
6. View sorted waiting list based on patient first name
7. View sorted history list based on sickness description
8. View sorted waiting list based on sickness description
9. Go Back to previous menu

Enter a number above: 7

Patient ID: PID1
Patient first Name: Hui Yin
Patient last Name: Loe
Patient Age: 18
Patient Gender: female
Patient Contact: 0193875637
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Fever
Medicine Information:
Disabled : 0
Visiting Day(dd/mm/yy) : 9/10/2021
Visiting Time(hour / min / sec) : 09:01:01
-----
Patient ID: PID4
Patient first Name: Kei Zhong
Patient last Name: Kong
Patient Age: 20
Patient Gender: male
Patient Contact: 0162895784
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Infection
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 13/10/2021
Visiting Time(hour / min / sec) : 13:00:00
-----
Patient ID: PID2
Patient first Name: Benson
Patient last Name: Junior
Patient Age: 17
Patient Gender: others
Patient Contact: 0173846758
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Migraine
Medicine Information:
Disabled : 1
```

*Figure 119 Doctor view on selecting option 7 in the viewing sub-menu*

The figure above shows the output of a sorted history list based on patient's sickness description. This output will only be shown when the user has selected option 7. The list that is displayed is sorted in ascending order based on the first alphabet of the patient's sickness description.

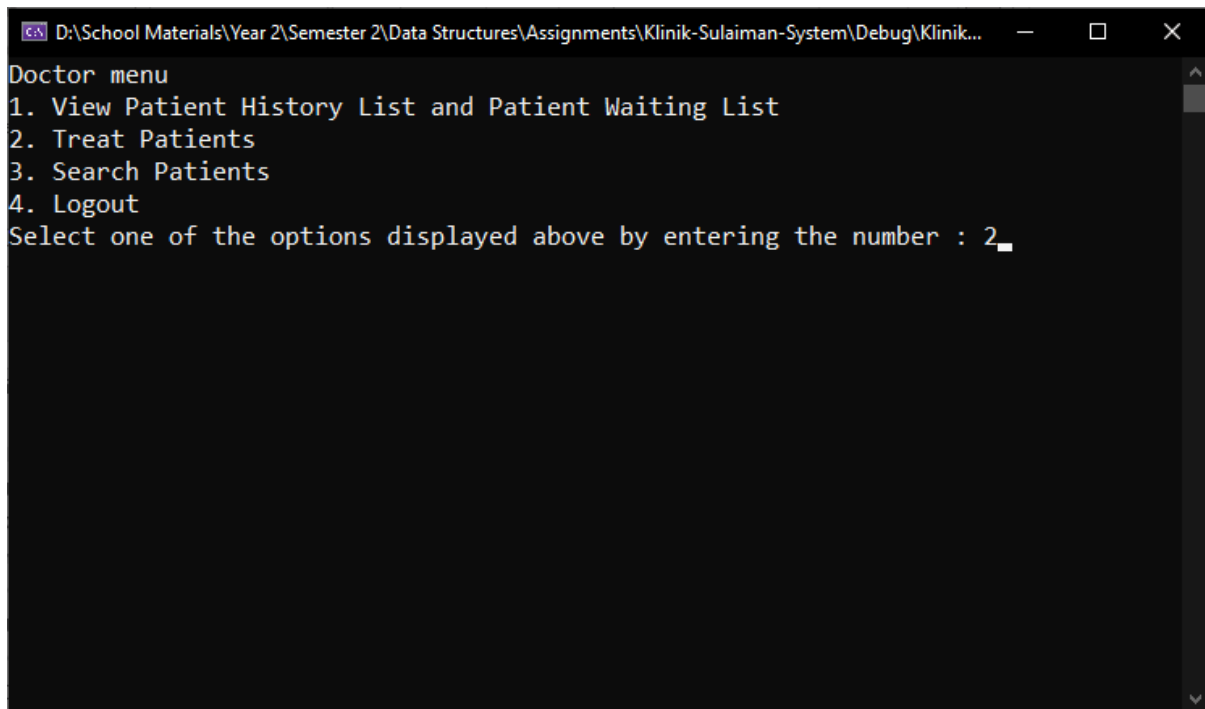
```
=== Doctor Viewing Menu ===
1. View entire patient waiting list
2. View patient waiting list in page by page mode
3. View entire patient visit history list (Descending order by visit time)
4. View patient visit history list in page by page mode
5. View sorted history list based on patient first name
6. View sorted waiting list based on patient first name
7. View sorted history list based on sickness description
8. View sorted waiting list based on sickness description
9. Go Back to previous menu

Enter a number above: 9
```

*Figure 120 User entering option 9 to go back to main menu*

Figure 120 shows the doctor can go back to main menu by selecting option 9.

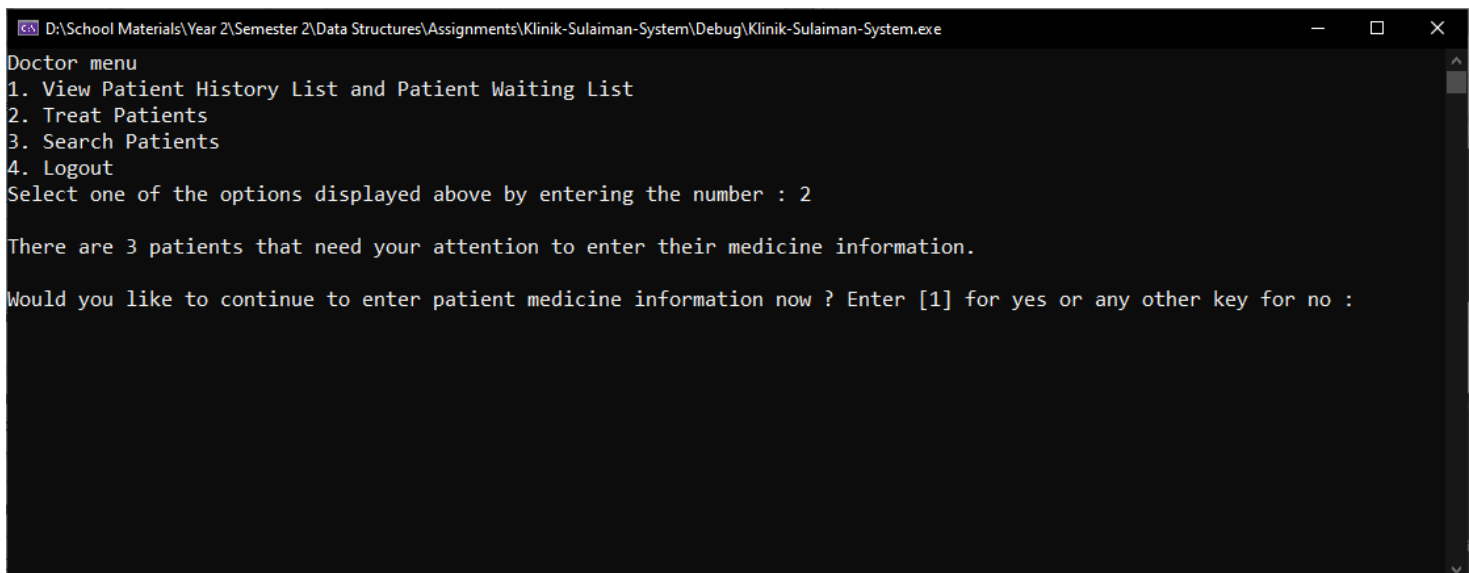




```
D:\School Materials\Year 2\Semester 2\Data Structures\Assignments\Klinik-Sulaiman-System\Debug\Klinik...
Doctor menu
1. View Patient History List and Patient Waiting List
2. Treat Patients
3. Search Patients
4. Logout
Select one of the options displayed above by entering the number : 2
```

*Figure 121 Doctor menu*

Figure 121 shows the treat patient option is selected.

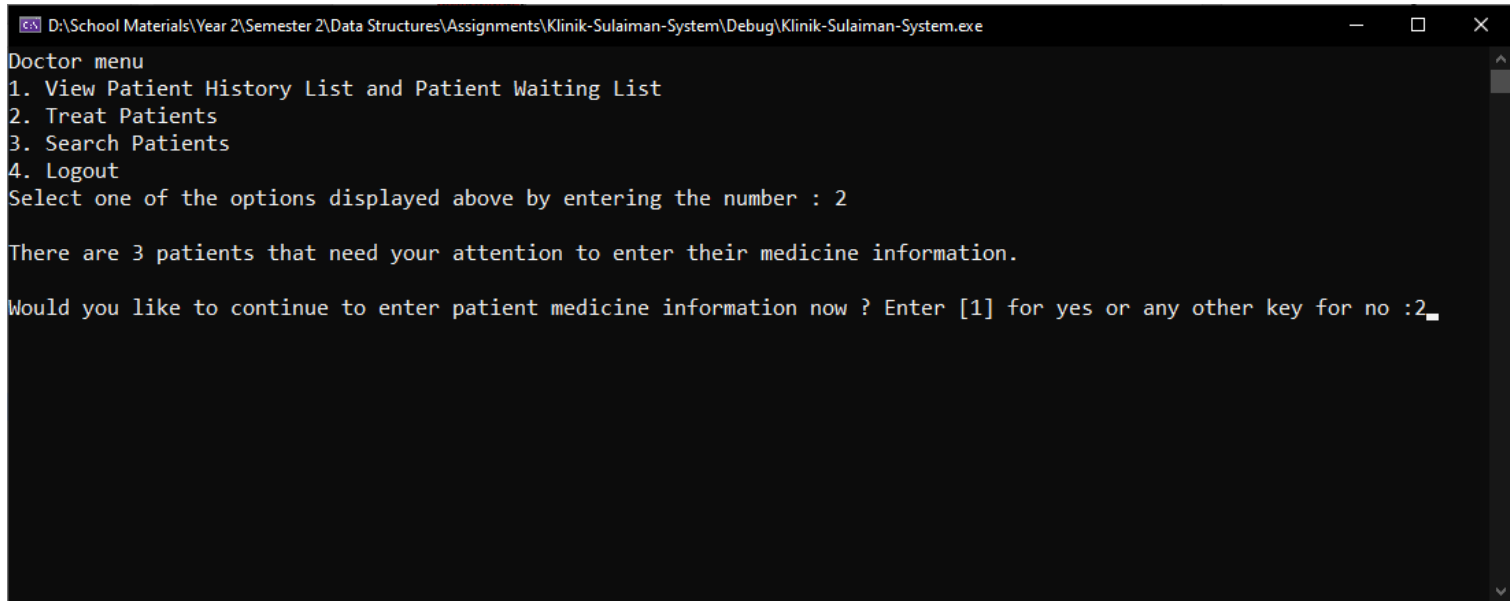


```
D:\School Materials\Year 2\Semester 2\Data Structures\Assignments\Klinik-Sulaiman-System\Debug\Klinik-Sulaiman-System.exe
Doctor menu
1. View Patient History List and Patient Waiting List
2. Treat Patients
3. Search Patients
4. Logout
Select one of the options displayed above by entering the number : 2

There are 3 patients that need your attention to enter their medicine information.
Would you like to continue to enter patient medicine information now ? Enter [1] for yes or any other key for no :
```

*Figure 122 Treat patients output*

Figure 122 shows a prompt asking the doctor user whether to proceed to enter the medical information for 3 patients that do not have any medical information registered. The two patients are under the logged-in doctor's supervision.



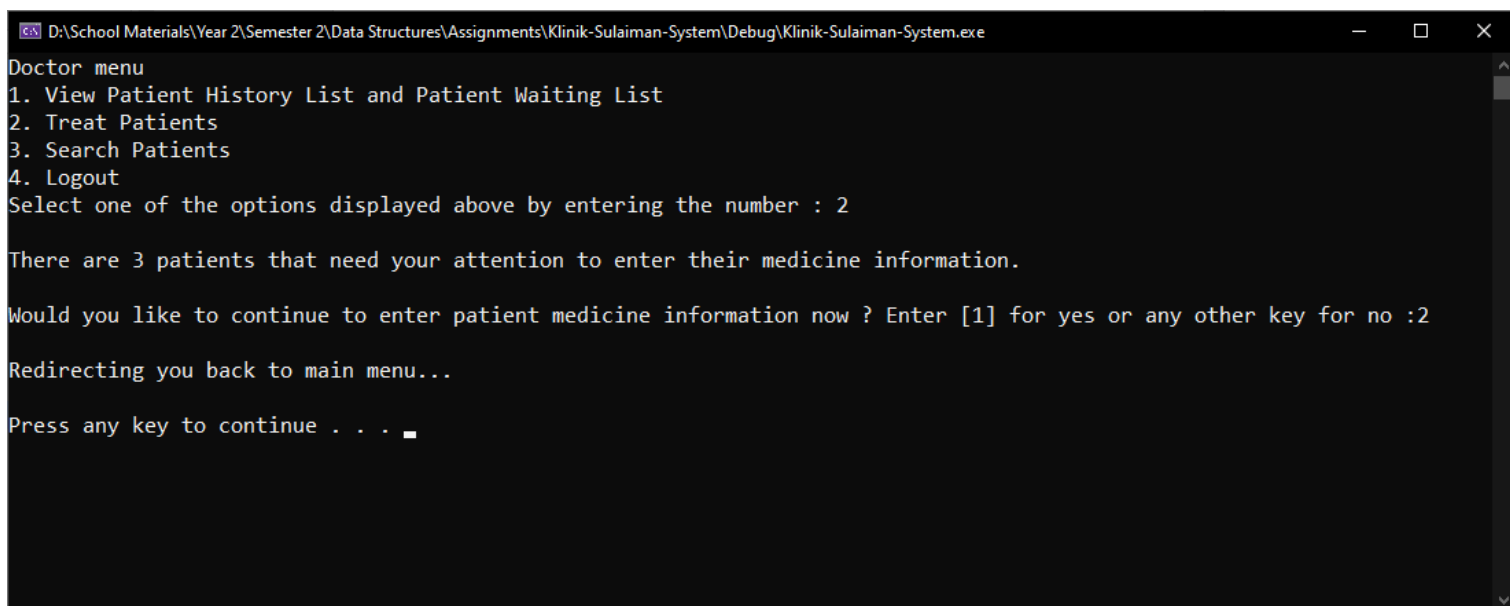
```
D:\School Materials\Year 2\Semester 2\Data Structures\Assignments\Klinik-Sulaiman-System\Debug\Klinik-Sulaiman-System.exe
Doctor menu
1. View Patient History List and Patient Waiting List
2. Treat Patients
3. Search Patients
4. Logout
Select one of the options displayed above by entering the number : 2

There are 3 patients that need your attention to enter their medicine information.

Would you like to continue to enter patient medicine information now ? Enter [1] for yes or any other key for no :2_
```

*Figure 123 Treat patients output screen*

Figure 123 shows the doctor user rejecting to input the medical information by entering 2.



```
D:\School Materials\Year 2\Semester 2\Data Structures\Assignments\Klinik-Sulaiman-System\Debug\Klinik-Sulaiman-System.exe
Doctor menu
1. View Patient History List and Patient Waiting List
2. Treat Patients
3. Search Patients
4. Logout
Select one of the options displayed above by entering the number : 2

There are 3 patients that need your attention to enter their medicine information.

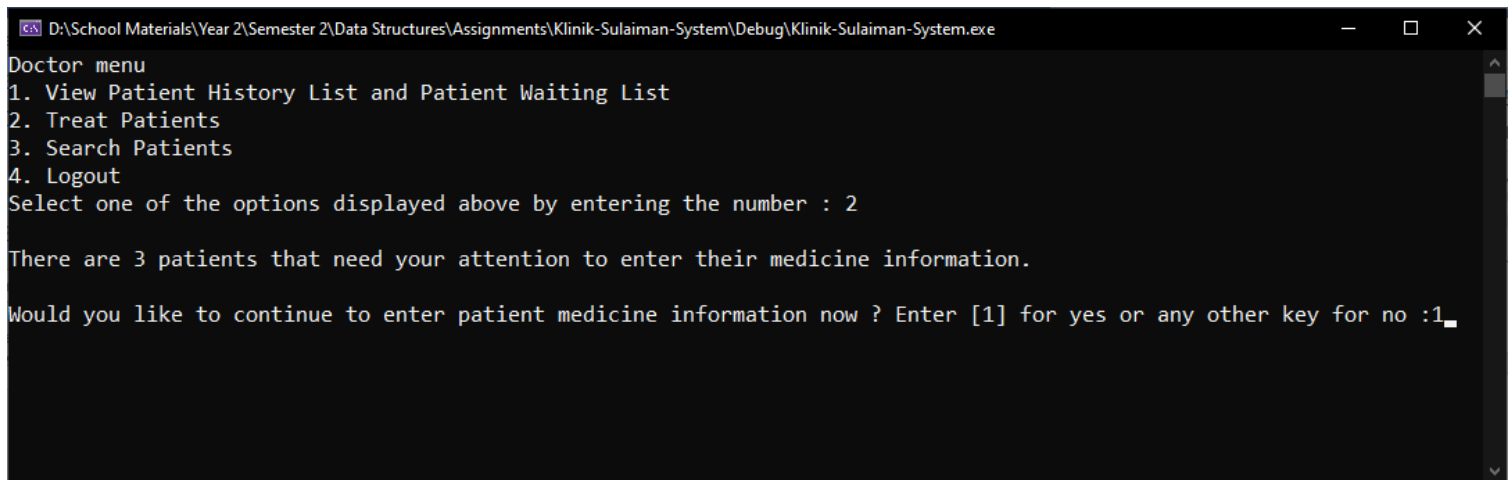
Would you like to continue to enter patient medicine information now ? Enter [1] for yes or any other key for no :2

Redirecting you back to main menu...

Press any key to continue . . . _
```

*Figure 124 Treat patients output screen*

Figure 124 shows the output screen before redirecting the doctor user back to the main menu, which is the doctor menu.



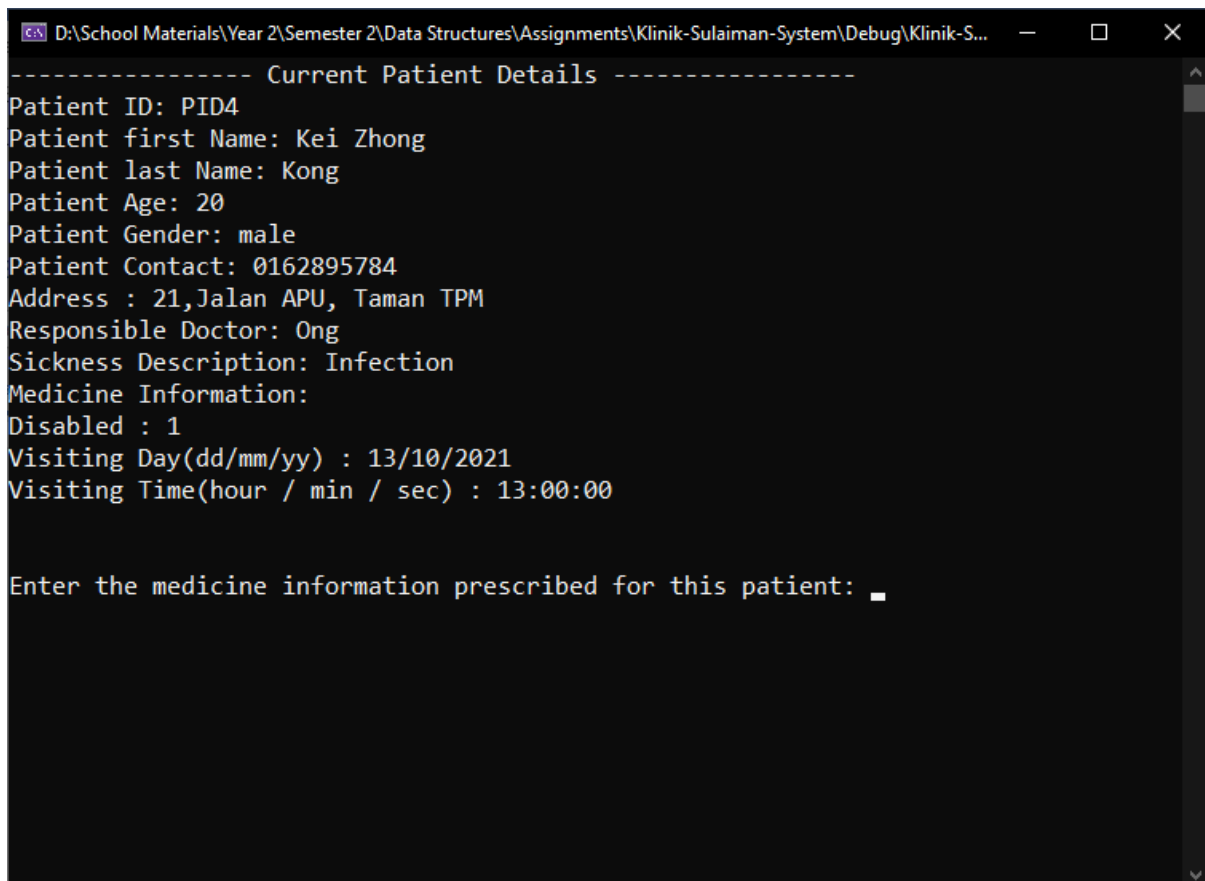
```
D:\School Materials\Year 2\Semester 2\Data Structures\Assignments\Klinik-Sulaiman-System\Debug\Klinik-Sulaiman-System.exe
Doctor menu
1. View Patient History List and Patient Waiting List
2. Treat Patients
3. Search Patients
4. Logout
Select one of the options displayed above by entering the number : 2

There are 3 patients that need your attention to enter their medicine information.

Would you like to continue to enter patient medicine information now ? Enter [1] for yes or any other key for no :1_
```

*Figure 125 Treat patients output screen*

Figure 125 shows the doctor accepting to input the medical information for the two patients by entering 1.

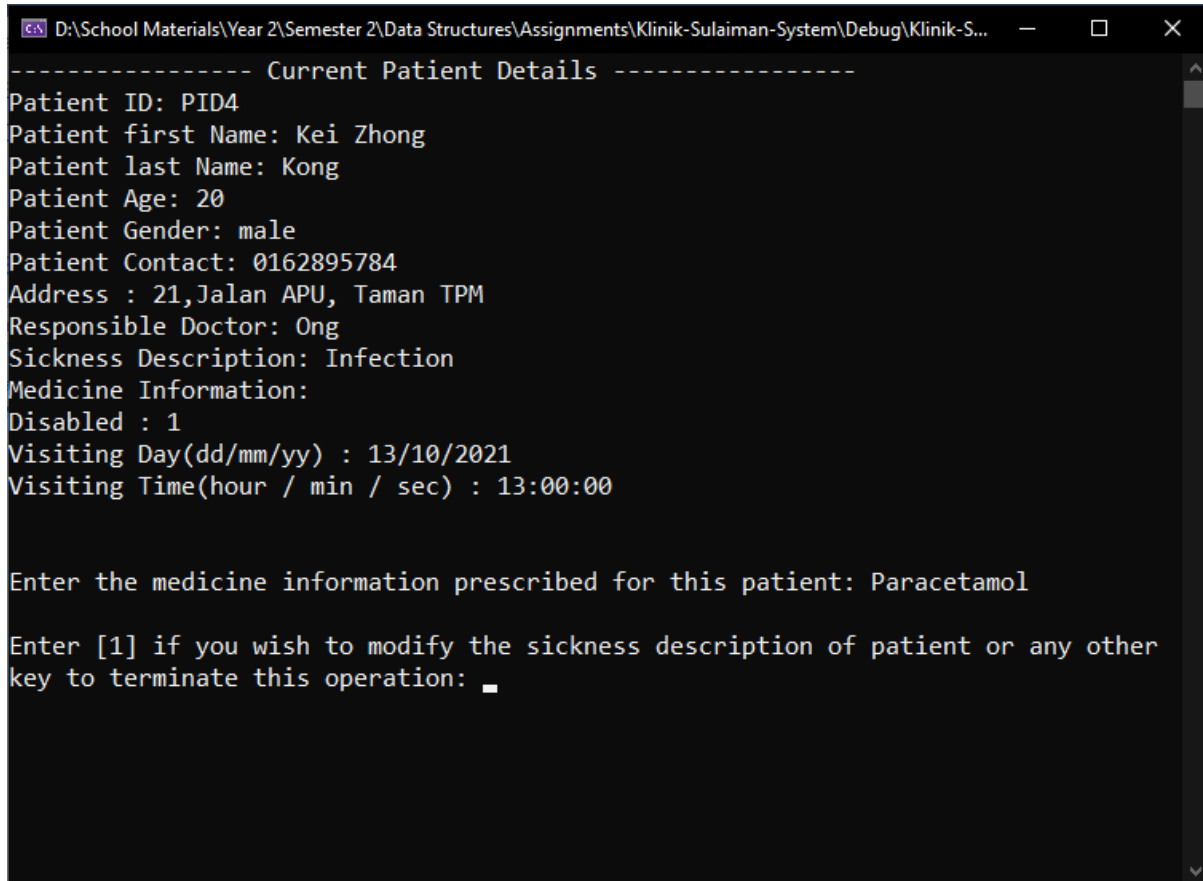


```
D:\School Materials\Year 2\Semester 2\Data Structures\Assignments\Klinik-Sulaiman-System\Debug\Klinik-S...
----- Current Patient Details -----
Patient ID: PID4
Patient first Name: Kei Zhong
Patient last Name: Kong
Patient Age: 20
Patient Gender: male
Patient Contact: 0162895784
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Infection
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 13/10/2021
Visiting Time(hour / min / sec) : 13:00:00

Enter the medicine information prescribed for this patient: _
```

*Figure 126 Patient to be treated output screen*

Figure 126 shows the output screen prompting the doctor to enter the medical information for the first patient, who is Kei Zhong.



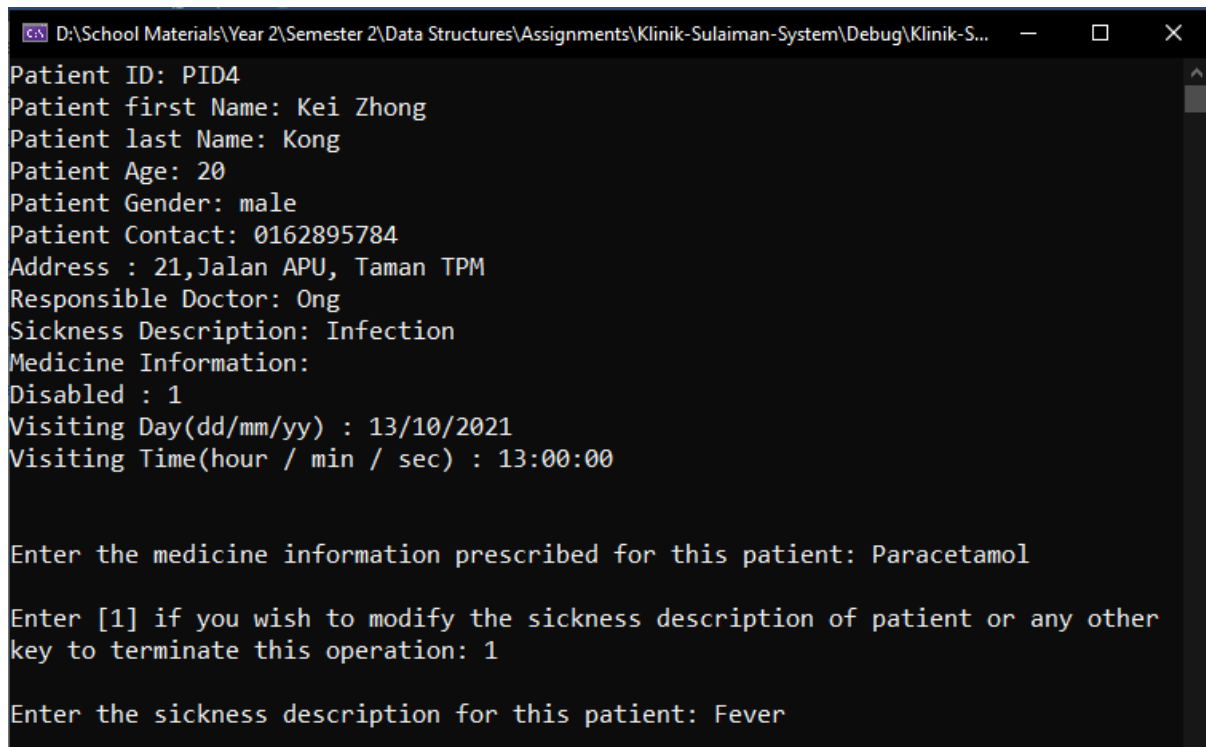
```
D:\School Materials\Year 2\Semester 2\Data Structures\Assignments\Klinik-Sulaiman-System\Debug\Klinik-S...
----- Current Patient Details -----
Patient ID: PID4
Patient first Name: Kei Zhong
Patient last Name: Kong
Patient Age: 20
Patient Gender: male
Patient Contact: 0162895784
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Infection
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 13/10/2021
Visiting Time(hour / min / sec) : 13:00:00

Enter the medicine information prescribed for this patient: Paracetamol

Enter [1] if you wish to modify the sickness description of patient or any other
key to terminate this operation: _
```

*Figure 127 Treating patient output screen*

Figure 127 shows the system prompting the doctor whether to change the sickness description after the medicine information is entered.



```
D:\School Materials\Year 2\Semester 2\Data Structures\Assignments\Klinik-Sulaiman-System\Debug\Klinik-S...
Patient ID: PID4
Patient first Name: Kei Zhong
Patient last Name: Kong
Patient Age: 20
Patient Gender: male
Patient Contact: 0162895784
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Infection
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 13/10/2021
Visiting Time(hour / min / sec) : 13:00:00

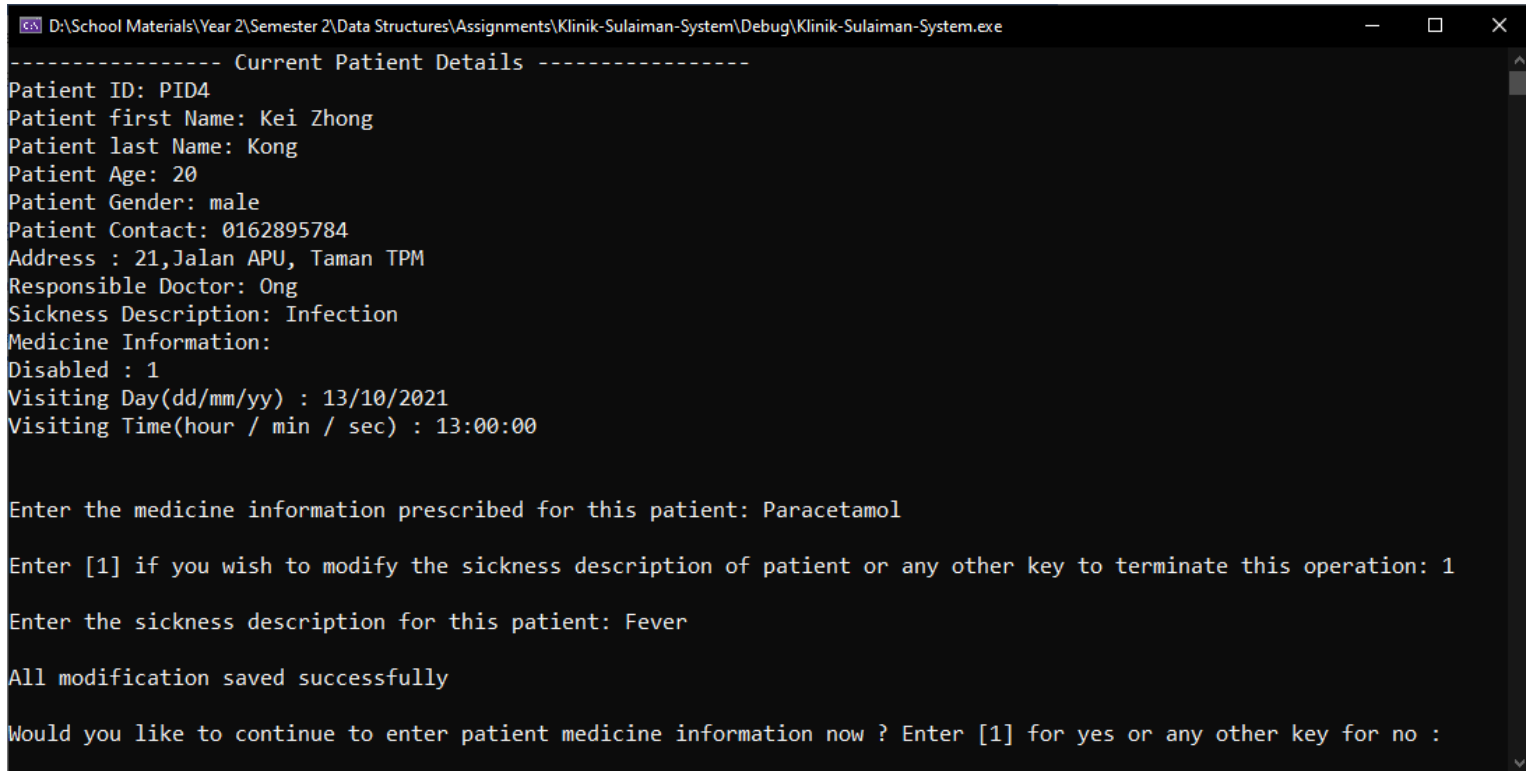
Enter the medicine information prescribed for this patient: Paracetamol

Enter [1] if you wish to modify the sickness description of patient or any other
key to terminate this operation: 1

Enter the sickness description for this patient: Fever
```

*Figure 128 Treating patient output screen*

Figure 128 shows the doctor has changed the sickness description for patient PID4 by entering 1. Patient PID4 new sickness description is fever.



```
D:\School Materials\Year 2\Semester 2\Data Structures\Assignments\Klinik-Sulaiman-System\Debug\Klinik-Sulaiman-System.exe
----- Current Patient Details -----
Patient ID: PID4
Patient first Name: Kei Zhong
Patient last Name: Kong
Patient Age: 20
Patient Gender: male
Patient Contact: 0162895784
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Infection
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 13/10/2021
Visiting Time(hour / min / sec) : 13:00:00

Enter the medicine information prescribed for this patient: Paracetamol

Enter [1] if you wish to modify the sickness description of patient or any other key to terminate this operation: 1

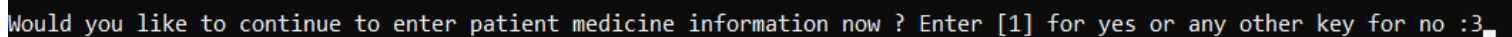
Enter the sickness description for this patient: Fever

All modification saved successfully

Would you like to continue to enter patient medicine information now ? Enter [1] for yes or any other key for no :
```

*Figure 129 Treating patient output screen*

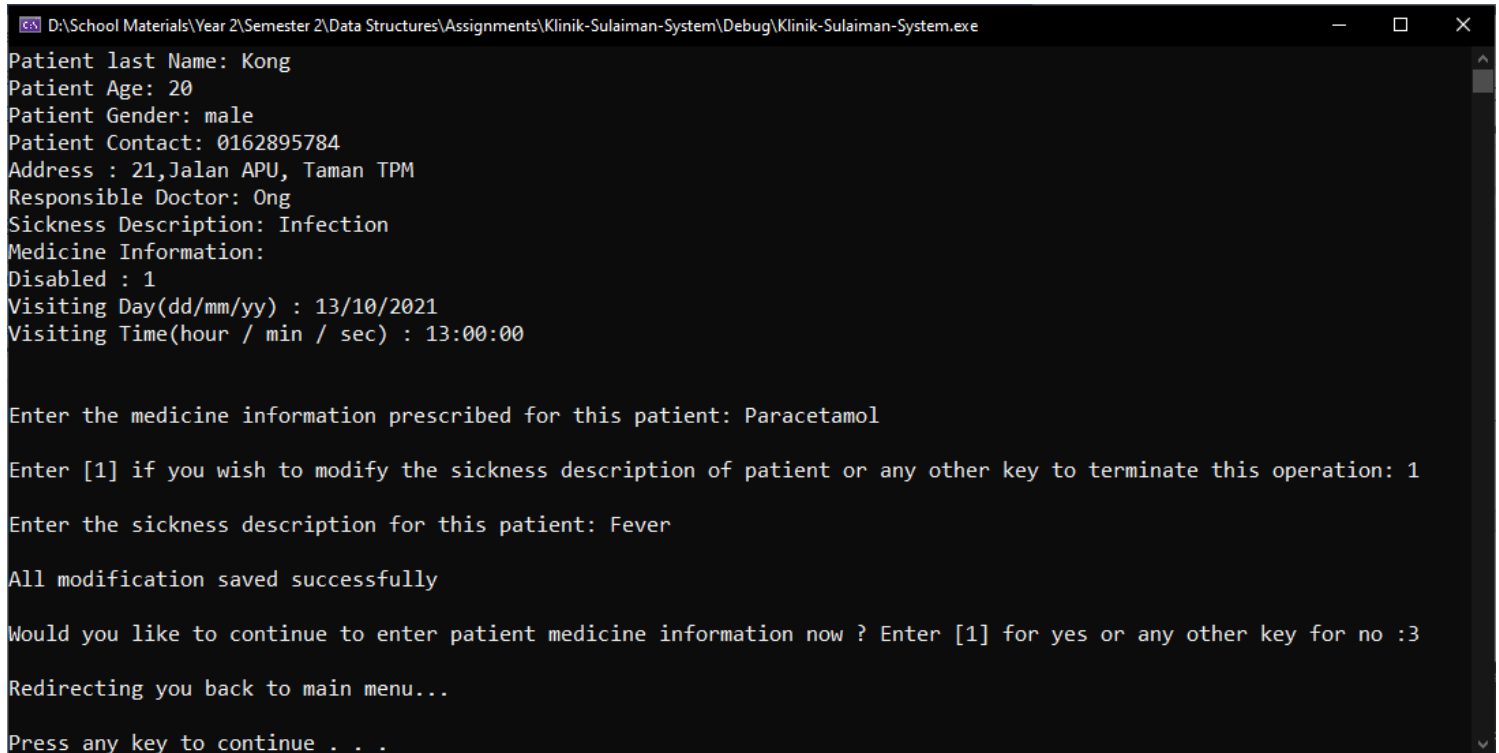
Figure 129 shows the output screen that all modifications are saved successfully. This implies that patient PID4 now has a registered medicine information and a changed sickness description. Afterwards, the system prompts the doctor whether to continue to enter medicine information for the rest of the patients that need the doctor's attention.



```
Would you like to continue to enter patient medicine information now ? Enter [1] for yes or any other key for no :3
```

*Figure 130 Treating patient output screen*

Figure 130 shows the doctor rejecting to continue to input medicine information for the rest of the patients by entering 3.



```
D:\School Materials\Year 2\Semester 2\Data Structures\Assignments\Klinik-Sulaiman-System\Debug\Klinik-Sulaiman-System.exe
Patient last Name: Kong
Patient Age: 20
Patient Gender: male
Patient Contact: 0162895784
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Infection
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 13/10/2021
Visiting Time(hour / min / sec) : 13:00:00

Enter the medicine information prescribed for this patient: Paracetamol

Enter [1] if you wish to modify the sickness description of patient or any other key to terminate this operation: 1

Enter the sickness description for this patient: Fever

All modification saved successfully

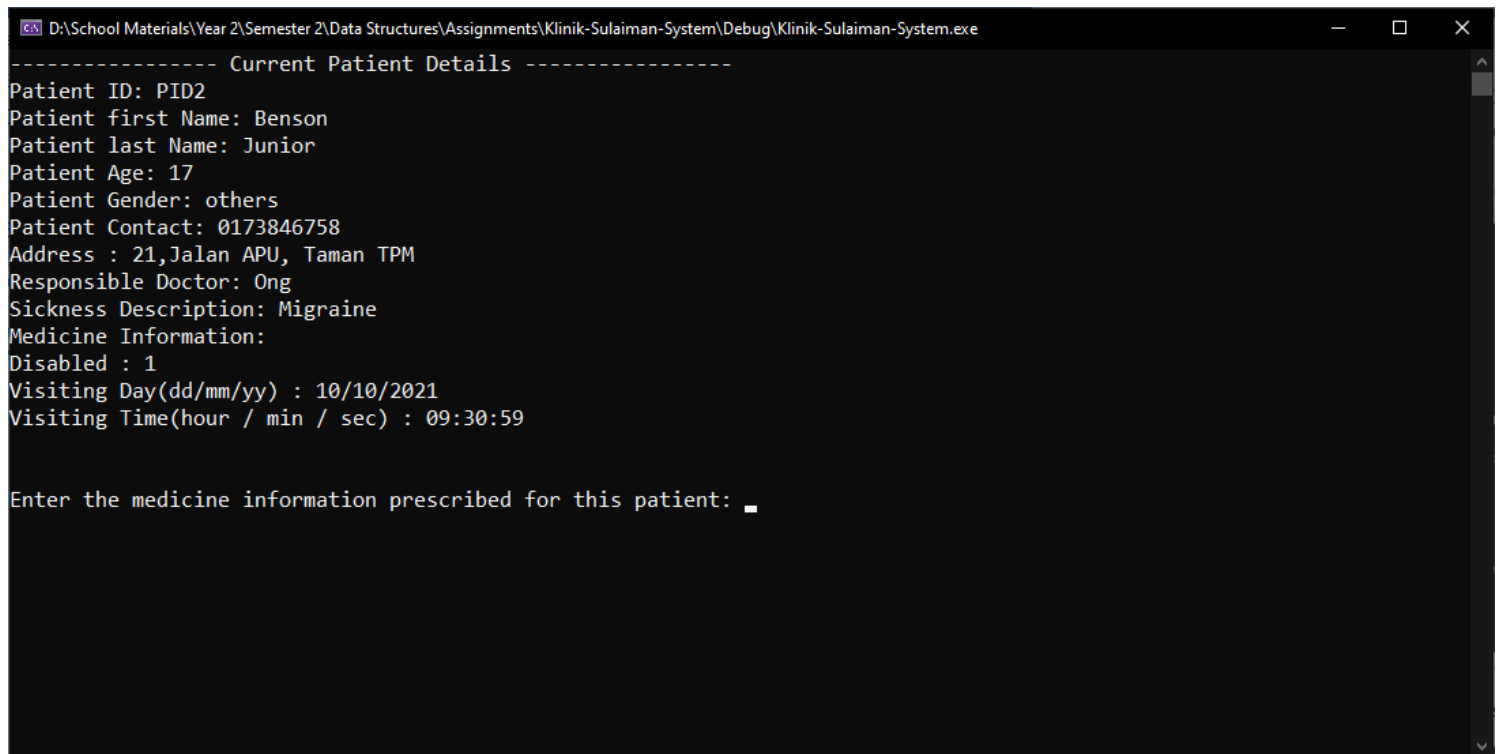
Would you like to continue to enter patient medicine information now ? Enter [1] for yes or any other key for no :3

Redirecting you back to main menu...

Press any key to continue . . .
```

*Figure 131 Treating patient output screen*

Figure 131 shows the output screen that redirects the doctor back to the main menu.



```
D:\School Materials\Year 2\Semester 2\Data Structures\Assignments\Klinik-Sulaiman-System\Debug\Klinik-Sulaiman-System.exe

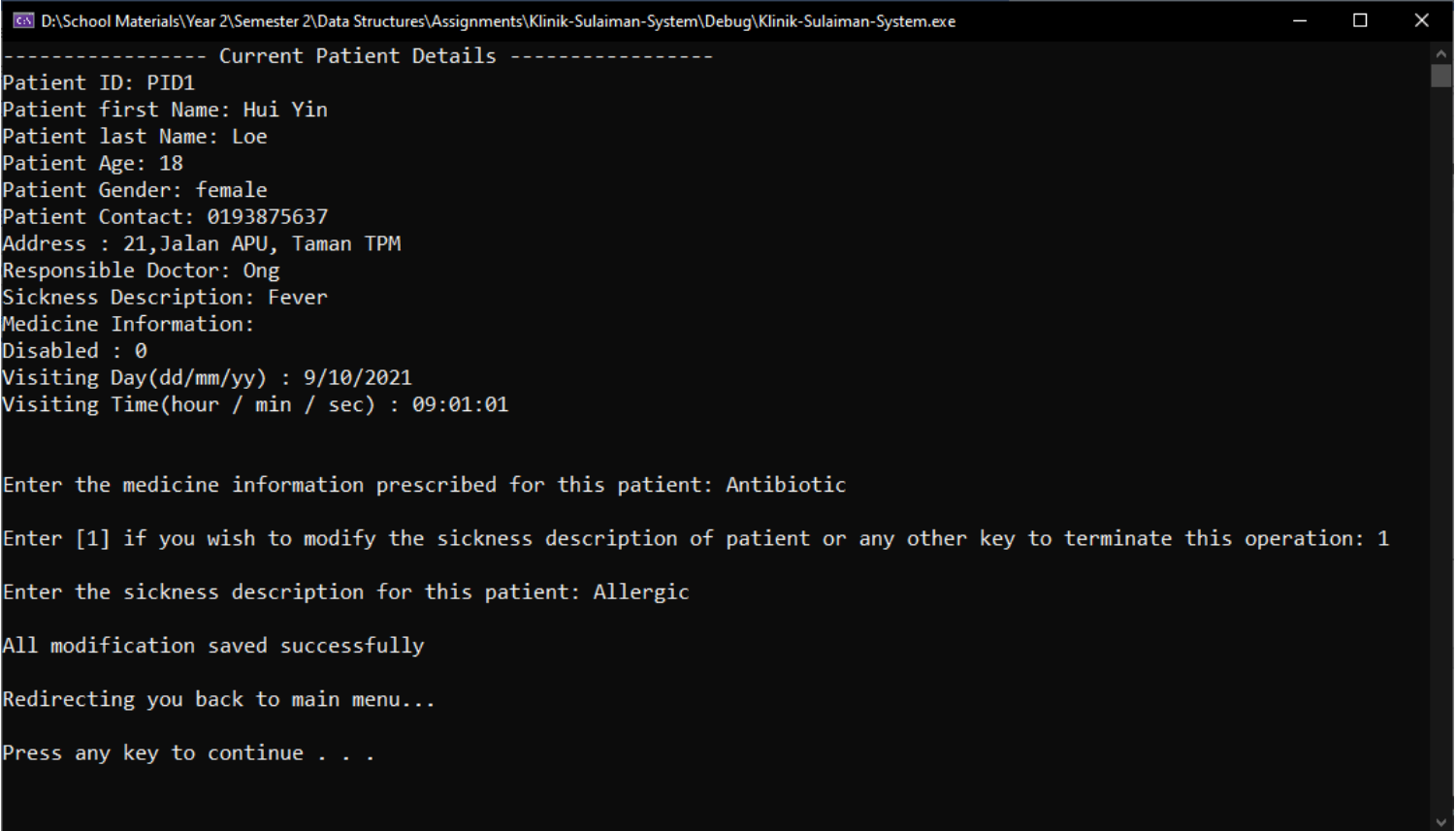
----- Current Patient Details -----
Patient ID: PID2
Patient first Name: Benson
Patient last Name: Junior
Patient Age: 17
Patient Gender: others
Patient Contact: 0173846758
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Migraine
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 10/10/2021
Visiting Time(hour / min / sec) : 09:30:59

Enter the medicine information prescribed for this patient: _
```

*Figure 132 Treating patient output screen*

Otherwise, Figure 132 shows the output screen that contains the details of the next patient that needs his or her medicine information to be entered by the doctor. Similarly, the doctor will be asked to enter the prescribed medicine information before deciding whether to change the patient's sickness description.





```
D:\School Materials\Year 2\Semester 2\Data Structures\Assignments\Klinik-Sulaiman-System\Debug\Klinik-Sulaiman-System.exe

----- Current Patient Details -----
Patient ID: PID1
Patient first Name: Hui Yin
Patient last Name: Loe
Patient Age: 18
Patient Gender: female
Patient Contact: 0193875637
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Fever
Medicine Information:
Disabled : 0
Visiting Day(dd/mm/yy) : 9/10/2021
Visiting Time(hour / min / sec) : 09:01:01

Enter the medicine information prescribed for this patient: Antibiotic

Enter [1] if you wish to modify the sickness description of patient or any other key to terminate this operation: 1

Enter the sickness description for this patient: Allergic

All modification saved successfully

Redirecting you back to main menu...

Press any key to continue . . .
```

*Figure 133 Treating patient output screen*

Figure 133 shows the output screen when there are no more patients who require the doctor's attention to input their prescribed medicine information. The system will automatically redirect the doctor back to the main menu.

```
Doctor menu
1. View Patient History List and Patient Waiting List
2. Treat Patients
3. Search Patients
4. Logout
Select one of the options displayed above by entering the number : 3
```

*Figure 134 Doctor menu*

Figure 134 shows the doctor selecting option 3 to search patients.

```
=== Doctor Searching Menu ===
1. Search by Patient ID
2. Search by Patient First Name
3. Search by Sickness Description
4. Medicine Information
5. Go back

Please select an option to search for the patients' profile:
```

*Figure 135 Doctor searching menu*

The figure above shows the patient searching menu that is accessible from the doctor's main menu. This search method displayed here is very similar to the one on the nurses' end, but with additional searching options, which are searching by sickness description and medicine information. Also, the doctor searching menu extends to searching for profiles of the patients from the visit history list on top of the patient waiting list with this method.

```
=== Doctor Searching Menu ===
1. Search by Patient ID
2. Search by Patient First Name
3. Search by Sickness Description
4. Medicine Information
5. Go back

Please select an option to search for the patients' profile: 6

WARNING: The input you have entered is not supported by the system, please select from the menu.

Press any key to continue . . .
```

*Figure 136 Doctor searching menu invalid input error message*

The figure above shows the error message in the doctor search menu when the user enters an invalid input.

```
=== Doctor Searching Menu ===
1. Search by Patient ID
2. Search by Patient First Name
3. Search by Sickness Description
4. Search by Medicine Information
5. Go back

Please select an option to search for the patients' profile: 1

Please enter the patient details to be searched with: PID1
Patient ID: PID1
Patient first Name: Hui Yin
Patient last Name: Loe
Patient Age: 18
Patient Gender: female
Patient Contact: 0193875637
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Fever
Medicine Information:
Disabled : 0
Visiting Day(dd/mm/yy) : 9/10/2021
Visiting Time(hour / min / sec) : 09:01:01
-----

Would you like to edit the information of any patient(s) displayed above ?
Enter [1] for yes or any other key for no :
```

*Figure 137 Doctor search by Patient ID output*

The figure above displays the output from the program by searching for patient ID “PID1” in the doctor search menu.

```
=== Doctor Searching Menu ===
1. Search by Patient ID
2. Search by Patient First Name
3. Search by Sickness Description
4. Search by Medicine Information
5. Go back

Please select an option to search for the patients' profile: 2

Please enter the patient details to be searched with: Chi En
Patient ID: PID8
Patient first Name: Chi En
Patient last Name: Chew
Patient Age: 21
Patient Gender: male
Patient Contact: 0193873275
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Wong
Sickness Description: Malnutrition
Medicine Information:
Disabled : 0
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 01:01:43
-----
Patient ID: PID3
Patient first Name: Chi En
Patient last Name: Ooi
Patient Age: 24
Patient Gender: female
Patient Contact: 0183758275
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Wong
Sickness Description: Nauseous
Medicine Information:
Disabled : 0
Visiting Day(dd/mm/yy) : 11/10/2021
Visiting Time(hour / min / sec) : 10:05:00
-----

Would you like to edit the information of any patient(s) displayed above ?
Enter [1] for yes or any other key for no :
```

*Figure 138 Doctor search by patient first name output*

The figure above shows the output from the doctors' searching menu by choosing the "Search by first name" option and entering "Chi En". The system prints the patient profiles with the identical first name passed into the program.

```
=== Doctor Searching Menu ===
1. Search by Patient ID
2. Search by Patient First Name
3. Search by Sickness Description
4. Search by Medicine Information
5. Go back

Please select an option to search for the patients' profile: 3

Please enter the patient details to be searched with: Migraine
Patient ID: PID7
Patient first Name: James
Patient last Name: Ethan
Patient Age: 30
Patient Gender: male
Patient Contact: 0124428984
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Migraine
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 01:01:43
-----
Patient ID: PID2
Patient first Name: Benson
Patient last Name: Junior
Patient Age: 17
Patient Gender: others
Patient Contact: 0173846758
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Migraine
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 10/10/2021
Visiting Time(hour / min / sec) : 09:30:59
-----

Would you like to edit the information of any patient(s) displayed above ?
Enter [1] for yes or any other key for no :
```

*Figure 139 Doctor search by patient sickness description output*

The figure above shows the output from searching for patients with the sickness description “Migraine”. It printed the details of 2 patients total, 1 of them from the patient waiting list and the other from the patient visit history lists.

```
=== Doctor Searching Menu ===
1. Search by Patient ID
2. Search by Patient First Name
3. Search by Sickness Description
4. Search by Medicine Information
5. Go back

Please select an option to search for the patients' profile: 4

Please enter the patient details to be searched with: Antibiotics
Patient ID: PID5
Patient first Name: Hua Long
Patient last Name: Lee
Patient Age: 35
Patient Gender: male
Patient Contact: 0163847182
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Wong
Sickness Description: Prostate disease
Medicine Information: Antibiotics
Disabled : 0
Visiting Day(dd/mm/yy) : 16/10/2021
Visiting Time(hour / min / sec) : 16:55:00
-----

Would you like to edit the information of any patient(s) displayed above ?
Enter [1] for yes or any other key for no :
```

*Figure 140 Doctor search by patient medicine information output*

The figure above displays the output from the doctor searching menu and searching via medicine information and entering “Antibiotics” as the search parameter.

```
=== Doctor Searching Menu ===
1. Search by Patient ID
2. Search by Patient First Name
3. Search by Sickness Description
4. Search by Medicine Information
5. Go back

Please select an option to search for the patients' profile: 1

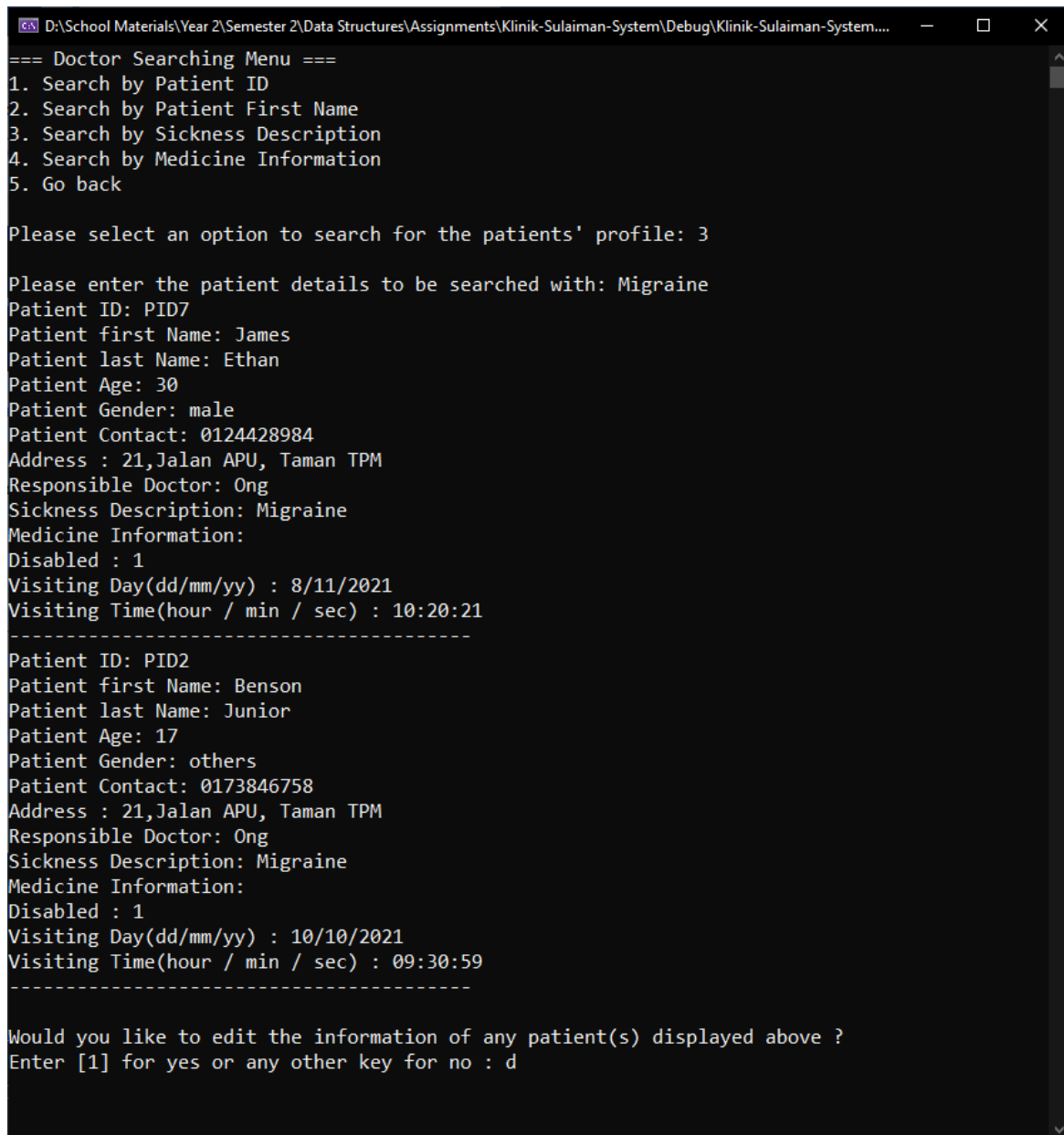
Please enter the patient details to be searched with: PID11

Patient(s) not found!

Press any key to continue . . .
```

*Figure 141 Doctor Search Menu*

The figure above displays the error message displayed to the user when a patient with the details passed in was not found in the waiting list and visit history list.



```
==== Doctor Searching Menu ====
1. Search by Patient ID
2. Search by Patient First Name
3. Search by Sickness Description
4. Search by Medicine Information
5. Go back

Please select an option to search for the patients' profile: 3

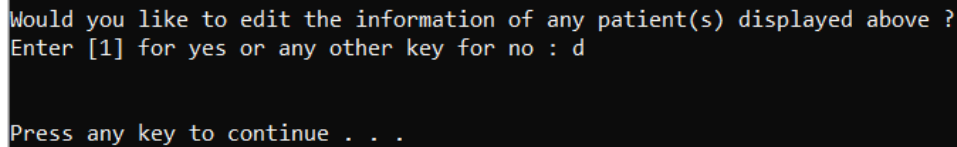
Please enter the patient details to be searched with: Migraine
Patient ID: PID7
Patient first Name: James
Patient last Name: Ethan
Patient Age: 30
Patient Gender: male
Patient Contact: 0124428984
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Migraine
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 10:20:21
-----
Patient ID: PID2
Patient first Name: Benson
Patient last Name: Junior
Patient Age: 17
Patient Gender: others
Patient Contact: 0173846758
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Migraine
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 10/10/2021
Visiting Time(hour / min / sec) : 09:30:59
-----

Would you like to edit the information of any patient(s) displayed above ?
Enter [1] for yes or any other key for no : d
```

*Figure 142 Search results*

Figure 142 shows the search results when the doctor wants to search for all patients with the sickness description of migraine. The doctor will also be prompted on whether to edit the patient information for the patients displayed above as shown in Figure 142.

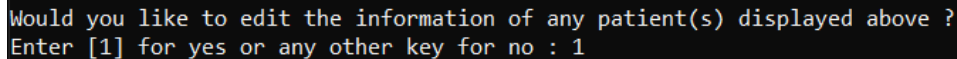




```
Would you like to edit the information of any patient(s) displayed above ?  
Enter [1] for yes or any other key for no : d  
  
Press any key to continue . . .
```

*Figure 143 Patient editor prompt*

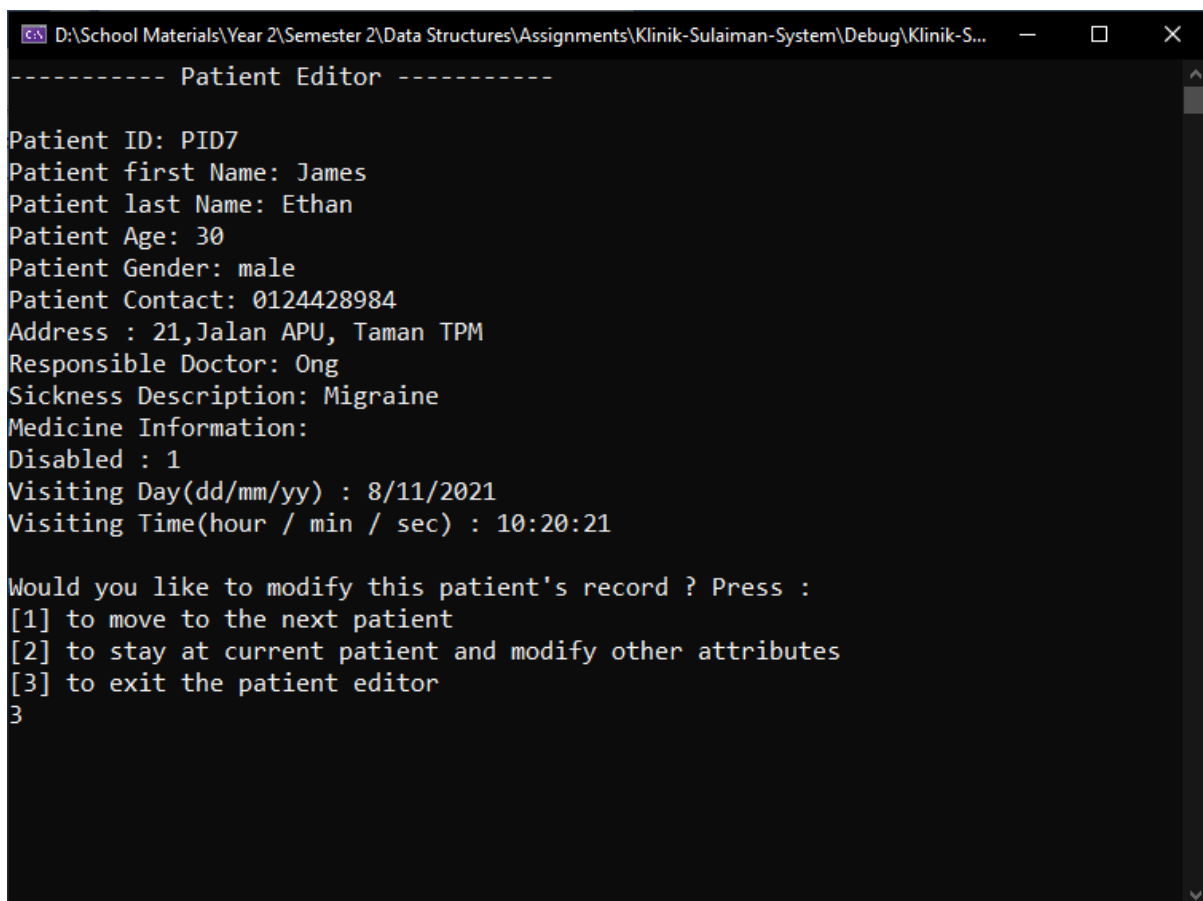
Figure 143 shows the output screen when the doctor rejects to edit any patient information for the patients returned by the search results. The doctor will be redirected back to the main menu.



```
Would you like to edit the information of any patient(s) displayed above ?  
Enter [1] for yes or any other key for no : 1
```

*Figure 144 Patient editor prompt*

On the other hand, the doctor can proceed to enter the patient editor by entering 1 for yes as shown in Figure 144.



```
D:\School Materials\Year 2\Semester 2\Data Structures\Assignments\Klinik-Sulaiman-System\Debug\Klinik-S...  
----- Patient Editor -----  
  
Patient ID: PID7  
Patient first Name: James  
Patient last Name: Ethan  
Patient Age: 30  
Patient Gender: male  
Patient Contact: 0124428984  
Address : 21,Jalan APU, Taman TPM  
Responsible Doctor: Ong  
Sickness Description: Migraine  
Medicine Information:  
Disabled : 1  
Visiting Day(dd/mm/yy) : 8/11/2021  
Visiting Time(hour / min / sec) : 10:20:21  
  
Would you like to modify this patient's record ? Press :  
[1] to move to the next patient  
[2] to stay at current patient and modify other attributes  
[3] to exit the patient editor  
3
```

*Figure 145 Patient editor*

Figure 145 shows the patient editor screen. In here, the doctor can traverse between the patients in the search results to modify them by entering the number that represents the command as shown in Figure 145.

```
Would you like to modify this patient's record ? Press :  
[1] to move to the next patient  
[2] to stay at current patient and modify other attributes  
[3] to exit the patient editor  
3  
  
Press any key to continue . . . █
```

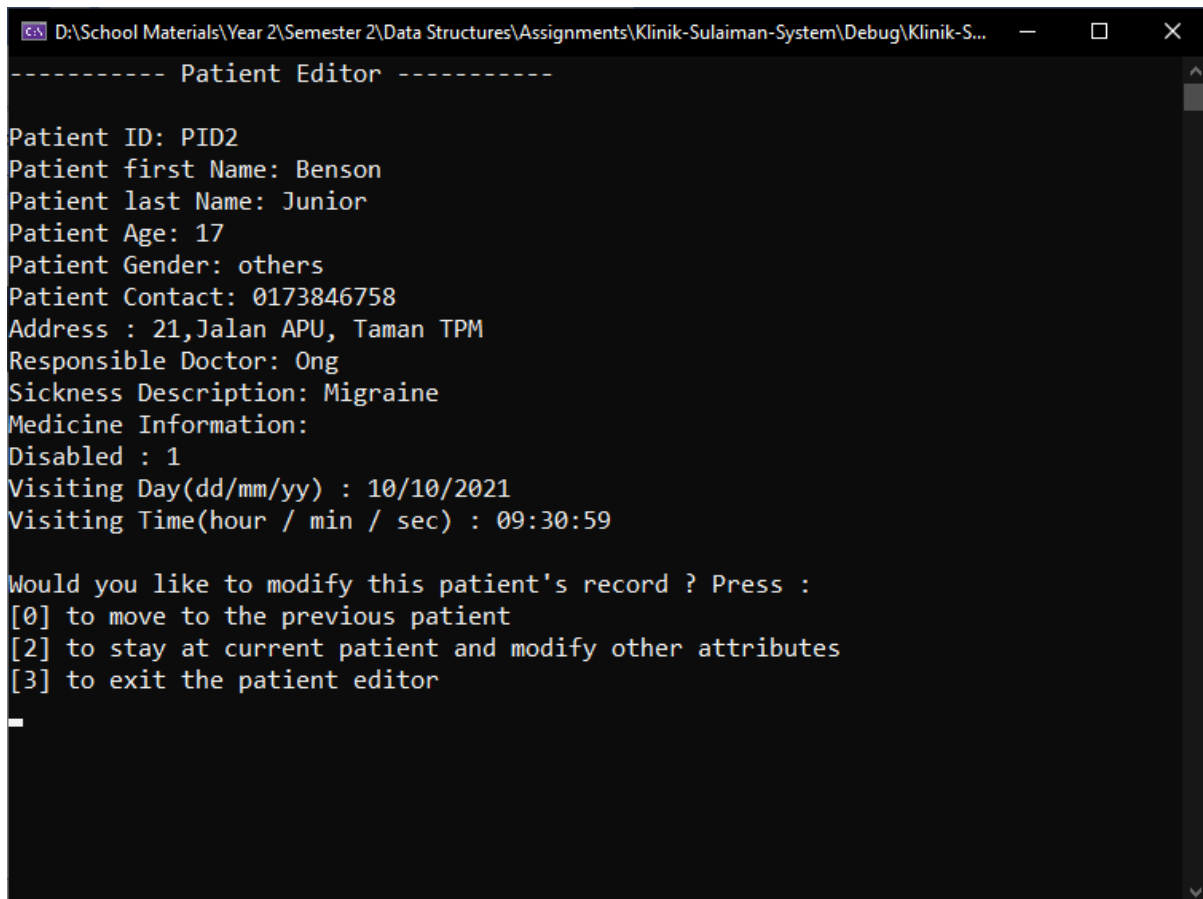
*Figure 146 Output screen*

Figure 146 shows the output screen when the doctor wants to exit the patient editor. The system will proceed to redirect the doctor back to the main menu.

```
D:\School Materials\Year 2\Semester 2\Data Structures\Assignments\Klinik-Sulaiman-System\Debug\Klinik-S...  -  □  X  
----- Patient Editor -----  
Patient ID: PID7  
Patient first Name: James  
Patient last Name: Ethan  
Patient Age: 30  
Patient Gender: male  
Patient Contact: 0124428984  
Address : 21,Jalan APU, Taman TPM  
Responsible Doctor: Ong  
Sickness Description: Migraine  
Medicine Information:  
Disabled : 1  
Visiting Day(dd/mm/yy) : 8/11/2021  
Visiting Time(hour / min / sec) : 10:20:21  
  
Would you like to modify this patient's record ? Press :  
[1] to move to the next patient  
[2] to stay at current patient and modify other attributes  
[3] to exit the patient editor  
1
```

*Figure 147 Patient editor*

Otherwise, the doctor can enter command 1 to move to the next patient in the patient editor as shown in Figure 147.

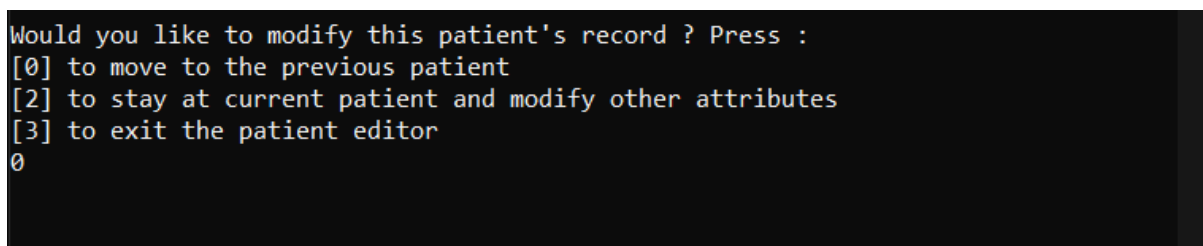


```
----- Patient Editor -----
Patient ID: PID2
Patient first Name: Benson
Patient last Name: Junior
Patient Age: 17
Patient Gender: others
Patient Contact: 0173846758
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Migraine
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 10/10/2021
Visiting Time(hour / min / sec) : 09:30:59

Would you like to modify this patient's record ? Press :
[0] to move to the previous patient
[2] to stay at current patient and modify other attributes
[3] to exit the patient editor
_
```

*Figure 148 Patient editor*

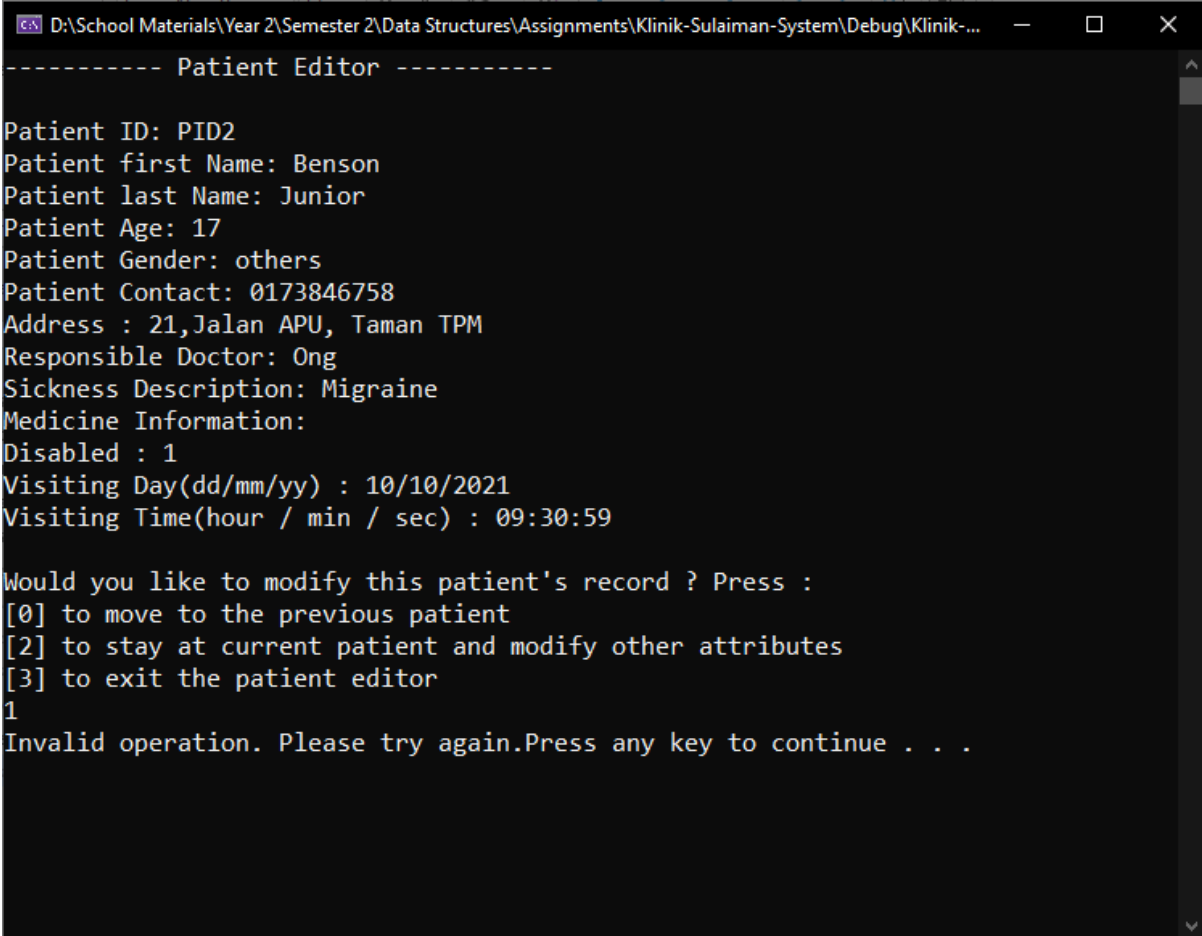
Figure 148 shows that the patient editor has switched to the next patient which is returned by the search results.



```
Would you like to modify this patient's record ? Press :
[0] to move to the previous patient
[2] to stay at current patient and modify other attributes
[3] to exit the patient editor
0
```

*Figure 149 Patient editor prompt*

In Figure 149, the doctor can choose to move back to the previous patient by entering command 0. There is no option for the doctor to move to the next patient as the current patient displayed is the last patient in the search results.

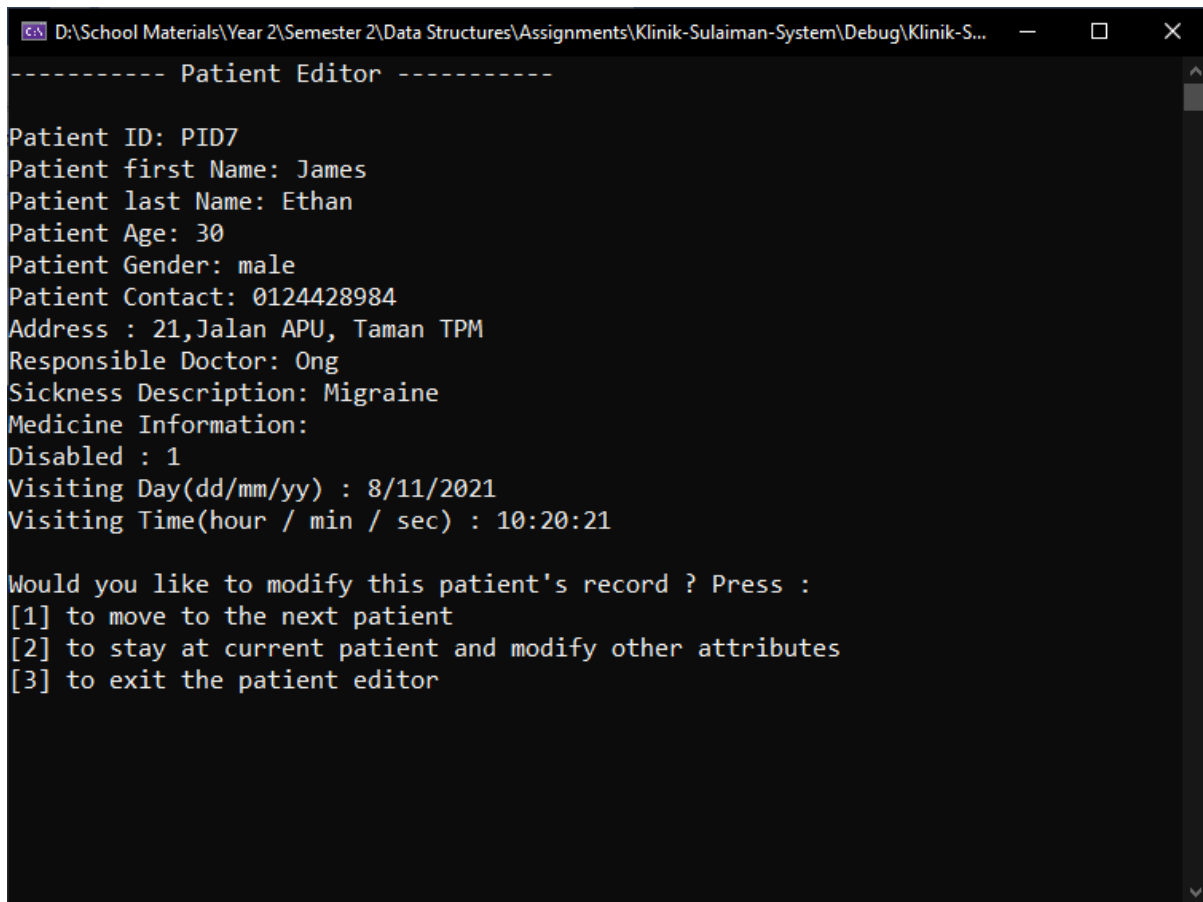


```
D:\School Materials\Year 2\Semester 2\Data Structures\Assignments\Klinik-Sulaiman-System\Debug\Klinik-...
----- Patient Editor -----
Patient ID: PID2
Patient first Name: Benson
Patient last Name: Junior
Patient Age: 17
Patient Gender: others
Patient Contact: 0173846758
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Migraine
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 10/10/2021
Visiting Time(hour / min / sec) : 09:30:59

Would you like to modify this patient's record ? Press :
[0] to move to the previous patient
[2] to stay at current patient and modify other attributes
[3] to exit the patient editor
1
Invalid operation. Please try again.Press any key to continue . . .
```

*Figure 150 Patient editor*

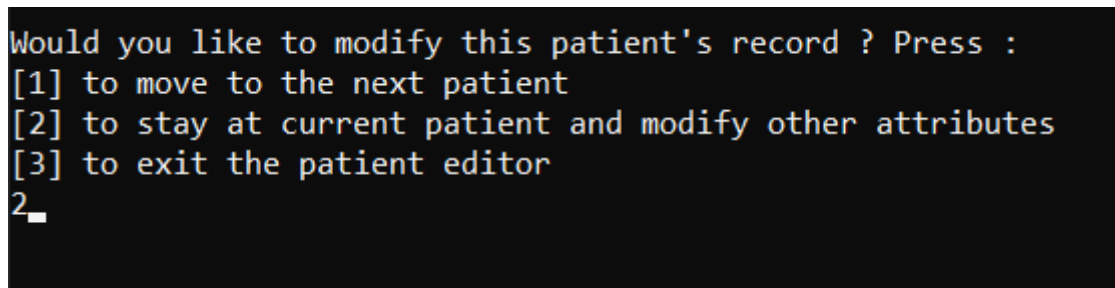
If the doctor enters an invalid command, then an error message is displayed to the users as shown in Figure 150. The doctor will have to try again and enter a valid command.



```
D:\School Materials\Year 2\Semester 2\Data Structures\Assignments\Klinik-Sulaiman-System\Debug\Klinik-S...  
----- Patient Editor -----  
Patient ID: PID7  
Patient first Name: James  
Patient last Name: Ethan  
Patient Age: 30  
Patient Gender: male  
Patient Contact: 0124428984  
Address : 21,Jalan APU, Taman TPM  
Responsible Doctor: Ong  
Sickness Description: Migraine  
Medicine Information:  
Disabled : 1  
Visiting Day(dd/mm/yy) : 8/11/2021  
Visiting Time(hour / min / sec) : 10:20:21  
  
Would you like to modify this patient's record ? Press :  
[1] to move to the next patient  
[2] to stay at current patient and modify other attributes  
[3] to exit the patient editor
```

*Figure 151 Patient editor*

Figure 151 shows the patient editor has switched back to the previous patient.



```
Would you like to modify this patient's record ? Press :  
[1] to move to the next patient  
[2] to stay at current patient and modify other attributes  
[3] to exit the patient editor  
2_
```

*Figure 152 Patient editor prompt*

If the doctor wishes to modify the current patient displayed by the patient editor, the doctor can enter command 2 as shown in Figure 152.

```
Press :  
[0] to change patient's first name  
[1] to change patient's last name  
[2] to change patient's age  
[3] to change patient's gender  
[4] to change patient's phone number  
[5] to change patient's address  
[6] to change patient's sickness description  
[7] to change patient's disability option  
[8] to change patient's doctor name  
[9] to change patient's medicine information  
Any other key to go back
```

*Figure 153 Patient editor prompt*

After the doctor has entered command 2, the system will proceed by prompting the doctor on which patient information to be changed. In Figure 153, the system only provides the doctor to modify the patient's information that is not generated by the system. Hence, the doctor cannot modify the patient ID, patient visit day, and patient visit time.

```
Press :  
[0] to change patient's first name  
[1] to change patient's last name  
[2] to change patient's age  
[3] to change patient's gender  
[4] to change patient's phone number  
[5] to change patient's address  
[6] to change patient's sickness description  
[7] to change patient's disability option  
[8] to change patient's doctor name  
[9] to change patient's medicine information  
Any other key to go back  
2  
Enter the patient's new age : █
```

*Figure 154 Patient editor prompt*

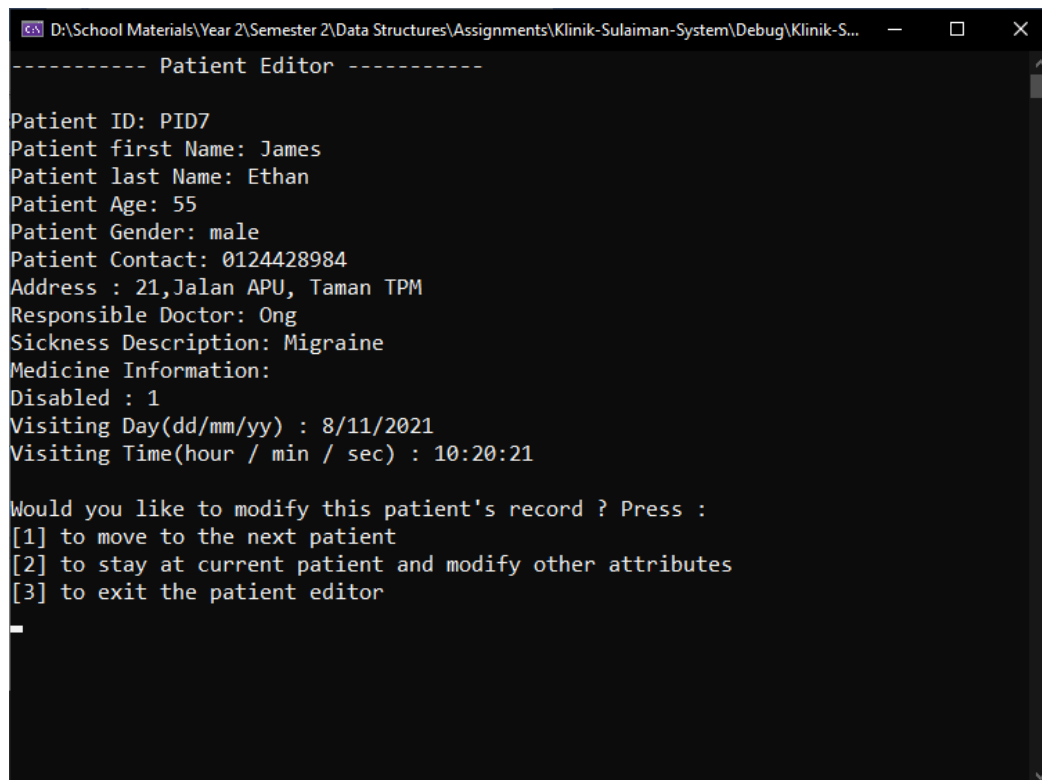
Figure 154 shows the output screen when the doctor attempts to modify the patient's age.

```
Enter the patient's new age : 55
All modifications are saved. Changes will be reflected on the next screen.

Press any key to continue . . .
```

*Figure 155 Output screen from patient editor prompt*

Figure 155 shows a successful message when the doctor enters a valid value for the patient's new age. Some of the attributes, such as age, gender, and phone number contain input validation that is similar to the output screen for the nurse when creating a patient.



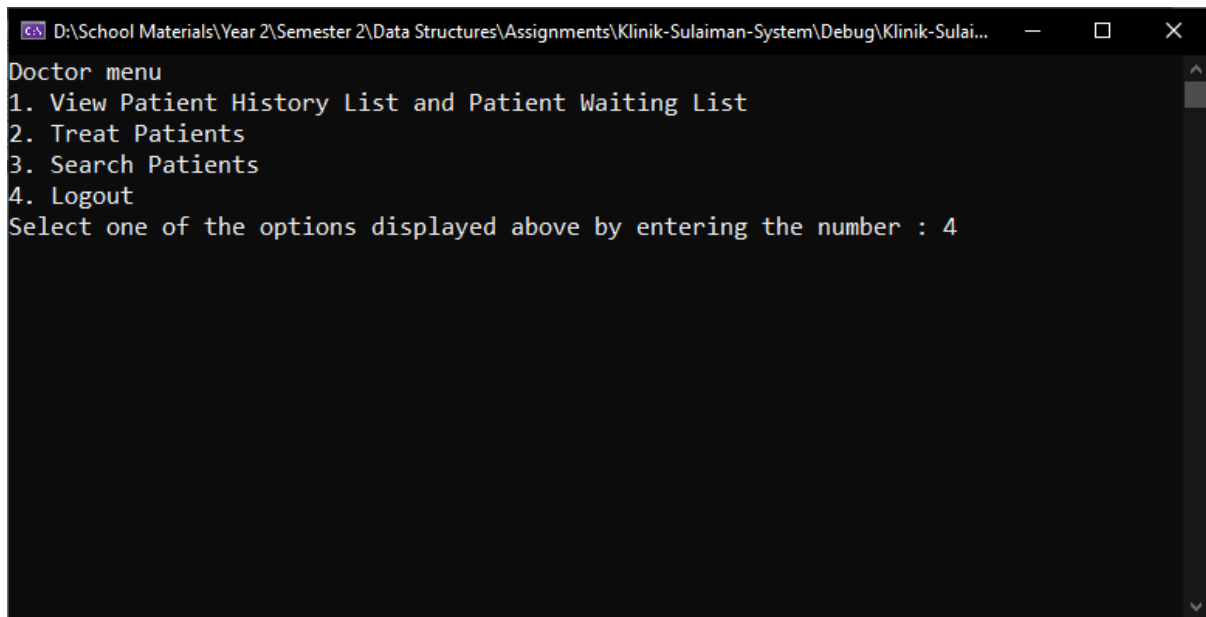
```
D:\School Materials\Year 2\Semester 2\Data Structures\Assignments\Klinik-Sulaiman-System\Debug\Klinik-S...
----- Patient Editor -----

Patient ID: PID7
Patient first Name: James
Patient last Name: Ethan
Patient Age: 55
Patient Gender: male
Patient Contact: 0124428984
Address : 21,Jalan APU, Taman TPM
Responsible Doctor: Ong
Sickness Description: Migraine
Medicine Information:
Disabled : 1
Visiting Day(dd/mm/yy) : 8/11/2021
Visiting Time(hour / min / sec) : 10:20:21

Would you like to modify this patient's record ? Press :
[1] to move to the next patient
[2] to stay at current patient and modify other attributes
[3] to exit the patient editor
-
```

*Figure 156 Patient editor*

Figure 156 shows the James Ethan has his age updated from 33 to 55 after performing the updating operation.



*Figure 157 Doctor menu*

Figure 157 shows the doctor can log out of the system by entering command 4 at the doctor menu. The system will then display the login page as shown in Figure 99.



## **Conclusion**

In conclusion, the system discussed the business flow of Klinik Sulaiman and how the system would aid and function within their business to help manage their business operations and data. The report details the attributes and functions of each class. The report also explained in detail the important roles of each class and their unique roles that they play in the system. Justifications were also made based on some of the decisions made such as the data structure selection, algorithm approaches as well as justification on menu design and system flow in order to seamlessly aid and benefit both users with very different job requirements into their daily business processes. The system is capable of facilitating both nurses and doctors and provides different functionalities based on their individual needs. Overall, the system developed fulfils all the requirements specified by Klinik Sulaiman such as adding patients and managing patient records with sorting functionalities. Besides, there are also additional features that are integrated into the system which are believed to be useful for Klinik Sulaiman.