**CPP104: LinkedList Operations Activity: Patient Management System**

**System Overview:**

The primary purpose of the patient management system is to streamline the administrative and medical operations within a healthcare facility. This system is designed to efficiently handle patient information, track medical histories, manage appointments, and ensure that healthcare providers have immediate access to crucial patient data. By automating and organizing patient records, the system aims to enhance the quality of care, reduce administrative burdens, and improve the overall patient experience.

The functionality of the system revolves around maintaining comprehensive and up-to-date records of all patients. This includes personal information, medical history, appointment schedules, treatment plans, and billing information. The system ensures that all data is stored securely and can be accessed and updated in real-time, providing healthcare providers with the most current information. Additionally, the system is designed to be user-friendly, allowing medical staff to navigate and utilize its features with ease.

The patient management system offers an intuitive interface for users to add new patient records, update treatment details, schedule appointments, and record new medical information. It also allows quick retrieval of records, search functionality for quick access, and updates to existing records to ensure accurate and current patient data, facilitating informed medical decisions.

The patient management system is a unique system that synchronizes patient data in real-time, protects sensitive information with robust security protocols, and is designed for ease of use. It features linked lists for easy scaling and powerful reporting tools for data-driven decision-making, improving healthcare outcomes and ensuring consistent performance despite large data volumes.

To sum up, the patient management system is an all-inclusive instrument intended to improve the efficacy and efficiency of healthcare management. Healthcare professionals can experience a seamless workflow thanks to the system's extensive features and user-friendly interface, which guarantees flexible and efficient management of patient records through linked list operations.

**Order Management Operations:**

* **Add Patient:** Adds a new patient to the linked list.
* **Serve Patient:** Removes the first patient from the linked list and marks them as served.
* **Current Patient:** Displays the patient currently being served.
* **Pending Patient List:** Displays all patients in the linked list.
* **Patient Capacity:** Checks if the list has reached its maximum capacity. The maximum number of patient for this program is 10 patients.

**User Manual:**

* Provide a step-by-step guide on how to implement and use the patient management system.
* Explain the initial setup and any required Python libraries.
* Detail the process for each patient management operation, including pseudocode and flowcharts if necessary.
* Include code snippets and explanations for key parts of the code.
* Show examples of patient management with initial and final states.

**Technical Documentation:**

* Describe the data structures used in the implementation of the linked list for patient management.

Linked list operations are crucial for efficient patient record management. These data structures consist of nodes, each containing data and a reference to the next. They handle dynamic data, update without reorganization, and simplify inserting or deleting operations. They allocate memory as needed for each new patient record, optimizing memory usage. Traversing a linked list is simple and efficient, allowing quick access to patient information.

* Include pseudocode outlining the logic of your patient management operations.

**Pseudocode**

Begin

Pass In: None

Class Node

Function \_\_init\_\_(self, data=None)

Set self.data to data

Set self.next to None

EndFunction

EndClass

Class LinkedList

Function \_\_init\_\_(self)

Set self.head to None

EndFunction

Function is\_empty(self)

Return True if self.head is None, otherwise return False

EndFunction

Function append(self, data)

Create new\_node with Node(data)

If is\_empty() is True

Set self.head to new\_node

Return

EndIf

Set last\_node to self.head

While last\_node.next is not None

Set last\_node to last\_node.next

EndWhile

Set last\_node.next to new\_node

EndFunction

Function delete\_node\_at\_position(self, position)

If self.head is None

Return

EndIf

Set temp to self.head

If position equals 0

Set self.head to temp.next

Set temp to None

Return

EndIf

For i from 0 to position - 1

Set temp to temp.next

If temp is None

Return

EndIf

EndFor

If temp is None or temp.next is None

Return

EndIf

Set next\_node to temp.next.next

Set temp.next to None

Set temp.next to next\_node

EndFunction

Function length(self)

Set count to 0

Set current to self.head

While current is not None

Increment count by 1

Set current to current.next

EndWhile

Return count

EndFunction

Function listHead(self)

If self.head is not None

Return self.head.data

Else

Set errorText to "There are no patients to serve yet"

Return errorText

EndIf

EndFunction

Function isFull(self)

Return length() >= 10

EndFunction

Function displayPatients(self)

Set current to self.head

Create empty list nodes

While current is not None

Append current.data to nodes

Set current to current.next

EndWhile

Return nodes

EndFunction

EndClass

Pass Out: None

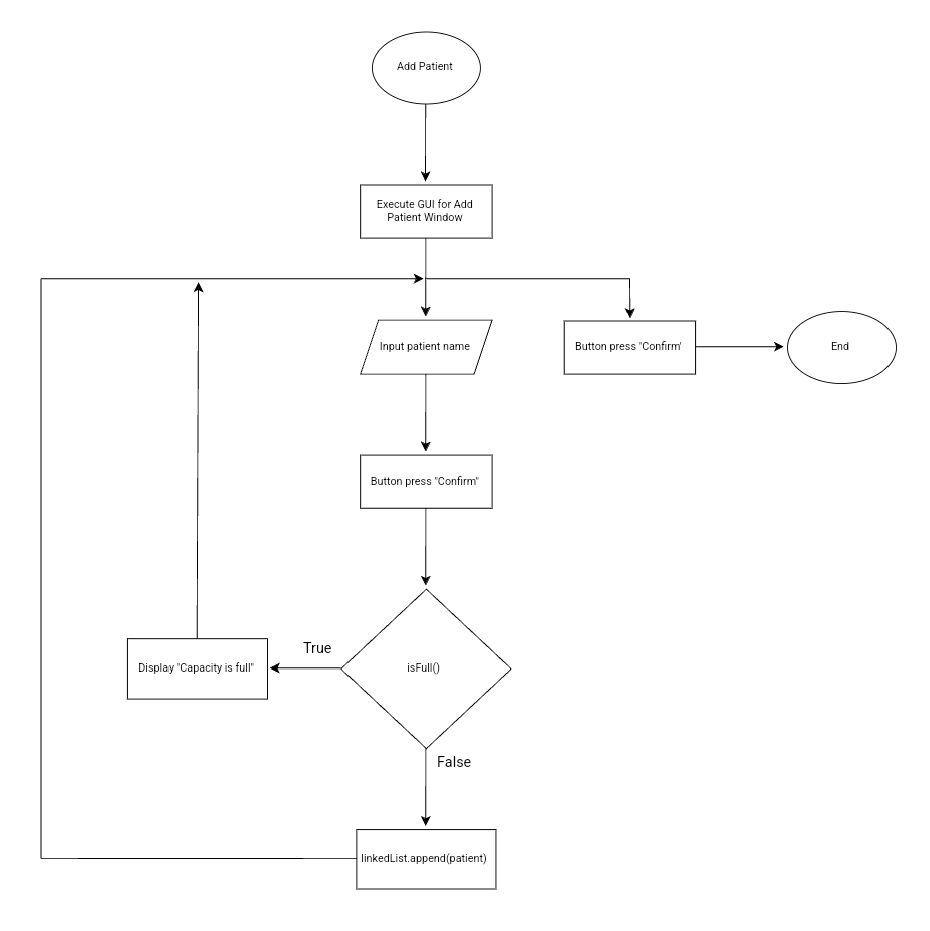
End

* Provide flowcharts to visualize the steps needed to implement each patient management operation.

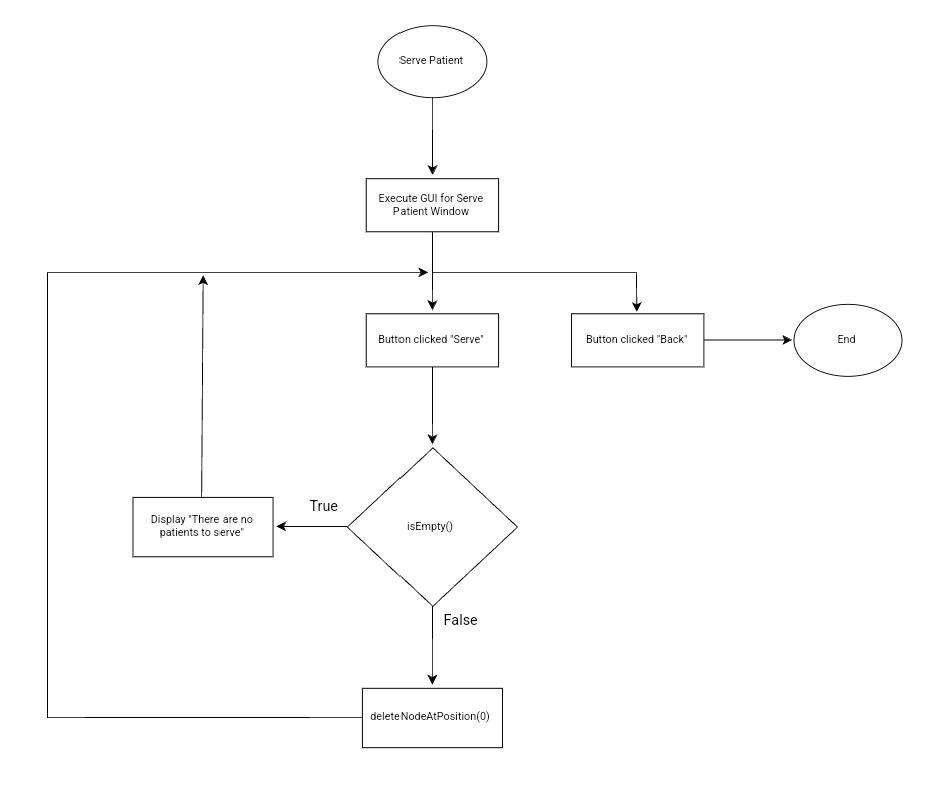
**Flowcharts**

Main

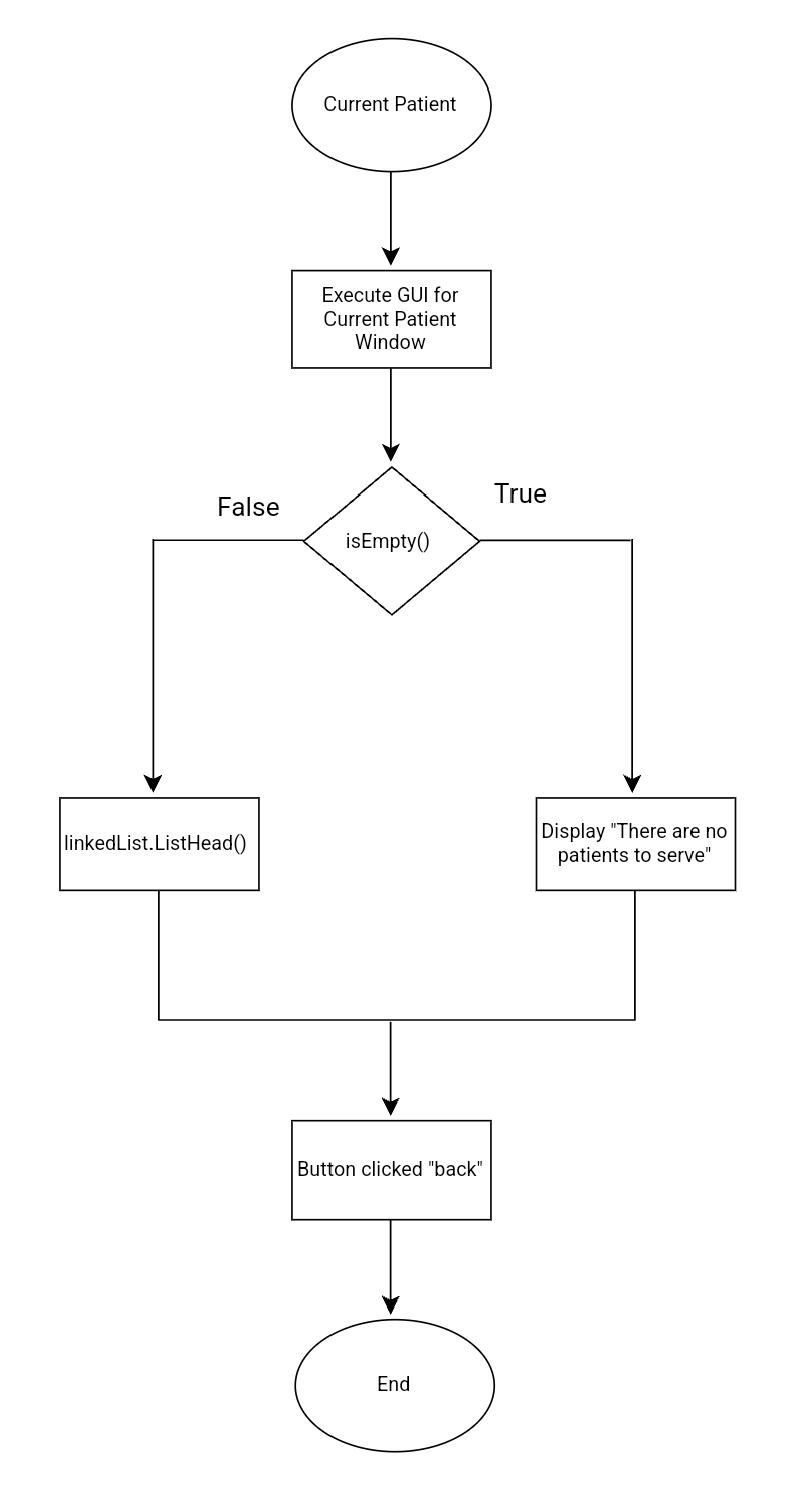
Add Patient



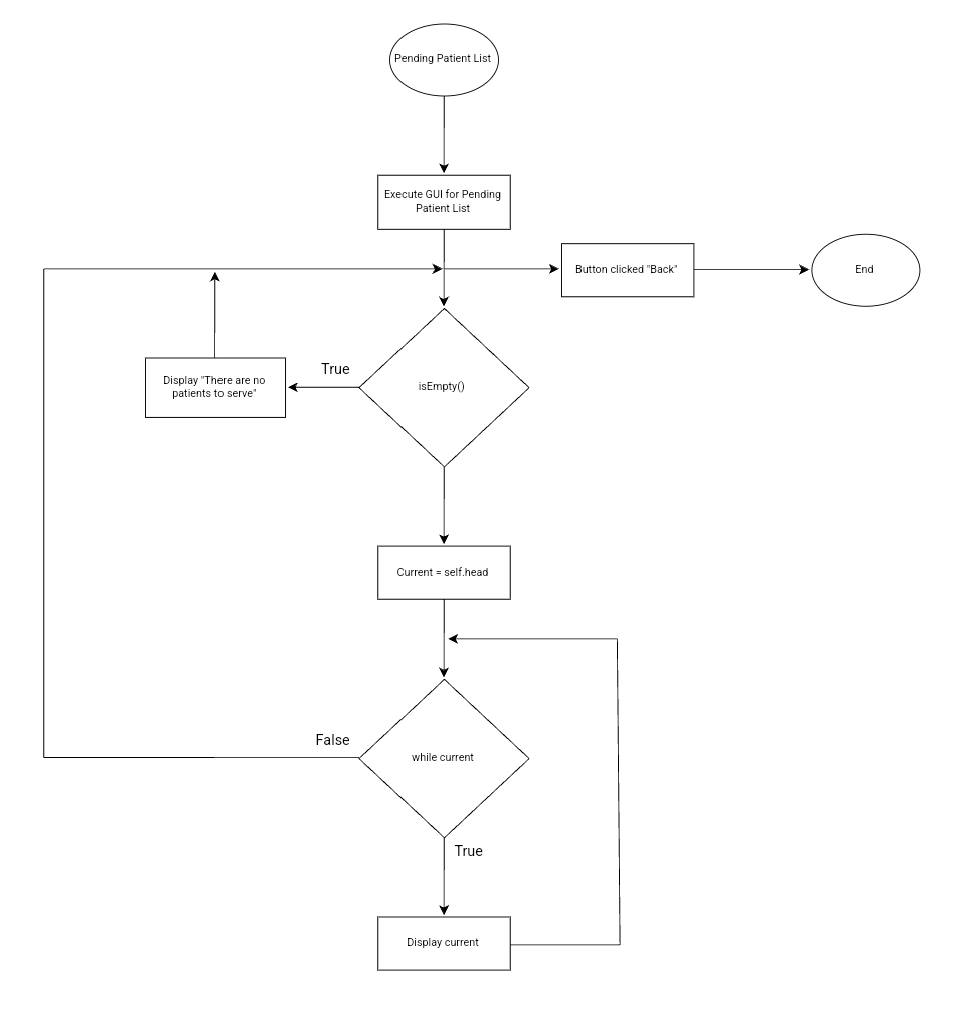
Serve Patient



Current Patient



Pending Patient List



**Known Issues and Limitations:**

In this program, the users are unable to search for a specific patient. They also cannot serve a specific patient if they need to prioritize serving the patient. The program can only serve the first patient added on the list. The developers recommend adding such feature in the future. This will help the users serving high priority patients.

**Future Enhancements:**

* Suggest potential improvements and additional features that could be added to the system in the future.

--- custom search of patients and serve custom patients

**Appendices:**

* Include any additional information or resources relevant to the patient management activity, such as sample input/output data or code snippets.
* If you use AI tools (e.g. ChatGPT) during the development process, include details on how they were utilized.
* List any online resources used, such as YouTube, StackOverflow, or other relevant websites.

**Members:**

|  |  |
| --- | --- |
| **Name** | **Contribution** |
| Donasco, Jomer John L. | Back-End Development and Documentation |
| Inay, Christian B. | Front-End Development and Documentation |
| Mañosca, Joph Anthony G. | Flowchart, Pseudocode, and Documentation |





