Specification Document

The following contains indepth information on specific features.

Navigaton

1. Selecting a Street (explorationMode)

Purpose : To allow the user to select a street from a menu, displaying available streets from the `streetData` vector and navigating accordingly.

Assumptions :

- The `streetData` vector contains multiple linked lists, each representing a street.
- Each street has a unique name.

Inputs :

- A vector of `StreetLinkedList` objects (`streetData`).

Outputs :

- CLI output of the selected street.

- Navigates to either the list of blocks in the selected street or displays all streets.

State Changes :

- No direct modification to the state of `streetData` or any linked list. However, the function passes control to the `blockMenu` or `printList` functions based on user input.

Cases & Expected Behavior :

- **Valid street selection** : Navigate to that street's block list.
- **"All" selected** : Prints all streets' data.
- **"Exit" selected** : Gracefully exits.
- **Invalid input** : Re-prompt the user until a valid input is provided.

2. Navigating Forward and Backward (streetExploration)

Purpose : To allow the user to navigate through the blocks (nodes) of a selected street, moving forward and backward through the nodes.

Assumptions :

- The linked list (street) has multiple blocks (nodes) to navigate.
- The current node is valid and correctly passed into the function.

Inputs :

- A `StreetLinkedList` object (list).
- A vector of linked lists (`streetData`).
- A pointer to the current node (`Node* current`).

Outputs :

- Displays the block's tree data or a message if the block is empty.
- CLI prompts for navigating to the next/previous node or returning to the menu.

State Changes :

- Moves the `current` pointer forward or backward in the linked list depending on user input.

- Optionally navigates back to `explorationMode` or `blockMenu`.

Cases & Expected Behavior :

- **'N' for next block** : Moves to the next node if available, otherwise alerts the user.
- **'P' for previous block** : Moves to the previous node if available, otherwise alerts the user.
- **'HEAD' or 'TAIL'** : Moves to the start or end of the list.
- **Invalid input** : Re-prompt the user until a valid input is given.

3. Printing All Lists in Geographic Representation (printList call in explorationMode)

Purpose : To print all linked lists (streets) and their blocks (nodes) in a geographic representation, showing cross streets and trees for each block.

Assumptions :

- Each linked list (street) is well-formed and contains blocks with relevant tree data.

Inputs :

- A vector of `StreetLinkedList` objects (`streetData`).

Outputs :

- CLI output representing the geographic data of each street (list) and block (node).

State Changes :

- None, since it only outputs data without modifying any structures.

Cases & Expected Behavior :

- **Empty lists** : No output for those streets, or a message indicating there are no blocks.
- **Non-empty lists** : Displays the tree data for each block on each street.

4. Graceful Exit (explorationMode with "Exit" option)

Purpose : To allow the user to exit the exploration mode in a graceful manner, showing a message before closing the program.

Assumptions :

- The user has navigated through the menu and decided to exit the program.

Inputs :

- A string representing the user's choice (`"Exit"`).

Outputs :

- CLI message: "Exiting...", followed by a pause (3 seconds).

State Changes :

- Terminates the exploration mode and exits the program or returns control to the main menu.

Cases & Expected Behavior :

- **"Exit" chosen** : Displays a message, waits, and exits.
- **Any other choice** : Continues the program.

5. Navigating to a Specific Block (blockMenu)

Purpose : To allow the user to select a block (node) on a specific street (list) and start navigating through that block's data.

Assumptions :

- The selected street contains multiple blocks.
- Each block is identified by its cross streets.

Inputs :

- A `StreetLinkedList` object (list).
- A vector of linked lists (`streetData`).

Outputs :

- CLI output showing the selected block or all blocks on the street.

State Changes :

- Passes control to `streetExploration` for navigating through blocks or `printList` to display all blocks.

Cases & Expected Behavior :

- **Valid block selection** : Navigates to that block's data.
- **"All" selected** : Displays all blocks for that street.
- **Invalid input** : Re-prompt until a valid selection is made.

Linked List Implementation

1. Insert (append & insertNode)

Purpose : To add new nodes to the linked list either at the end (`append`) or at any arbitrary position (`insertNode`).

Assumptions :

- The list can be empty when inserting a node.
- When inserting at a position, valid positions should be within the list's bounds or at the end.

Inputs :

- `append` takes a `Node&` (new node to be added) as input.
- `insertNode` takes a `Node&` (new node) and an `int` (position) as input.

Outputs :

- No return value.
- Node is added to the list.

State Changes :

- The linked list's structure changes by adding a new node.

- `head` and `tail` pointers are updated if inserting at the beginning or end of the list.

Cases & Expected Behavior :

- **Empty list** : Inserting a node should set both `head` and `tail` to the new node.
- **Insert at the head** : New node becomes the `head`.
- **Insert at the tail** : New node becomes the `tail`.
- **Insert in the middle** : Adjusts the pointers of surrounding nodes to include the new node.

2. Delete (deleteNode)

Purpose : To remove a node from the list based on the streets (approaching and leaving) associated with the node.

Assumptions :

- The list may contain zero or more nodes.
- The streets used to identify a node must be unique per node.

Inputs :

- `deleteNode` takes two `std::string&` parameters: `approachingStreet` and `leavingStreet`.

Outputs :

- No return value.
- The node is removed from the list or a message is printed if no match is found.

State Changes :

- Node pointers (`prev` and `next`) of surrounding nodes are updated.
- If the deleted node is `head` or `tail`, these pointers are adjusted.

Cases & Expected Behavior :

- **Empty list** : Print "Node not found".
- **Deleting the head** : Adjust `head` pointer.
- **Deleting the tail** : Adjust `tail` pointer.
- **Deleting a middle node** : Adjust the `prev` and `next` pointers of surrounding nodes.

- **Node not found** : Print a message.

3. Printing (printList)

Purpose : To display the details of each node in the linked list, including the number of trees and their distances for each block.

Assumptions :

- The list can contain zero or more nodes.
- Each node contains a `std::vector` of trees.

Inputs :

- No input parameters are taken.

Outputs :

- Outputs the details of each node and its trees to the console.

State Changes :

- No state change. The list remains unmodified.

Cases & Expected Behavior :

- **Empty list** : No output, as the loop doesn't execute.
- **Non-empty list** : Prints the details for each node in the list.

4. Search (searchNode)

Purpose : To find and return a node based on its associated cross streets (approaching and leaving streets).

Assumptions :

- The list can be empty or contain multiple nodes.
- Cross streets must uniquely identify a node.

Inputs :

- Two `std::string&` parameters: `approachingStreet` and `leavingStreet`.

Outputs :

- Returns a `Node*` pointer to the node if found, or `nullptr` if the node is not in the list.

State Changes :

- No state change. The list remains unmodified.

Cases & Expected Behavior :

- **Empty list** : Print "Node not found" and return `nullptr`.
- **Node found** : Return pointer to the matching node.
- **Node not found** : Print "Node not found" and return `nullptr`.