**Specifications**

**get\_head()**

* **Purpose:** Returns a pointer to the head node.
* **Assumptions:** List has been initialized.
* **Inputs:** None.
* **Outputs*:*** *Node\** pointing to the first node, or nullptr if list is empty.
* **State Changes:** None.
* **Cases and Expected Behavior:**
  + Returns nullptr for empty list.
  + Returns valid pointer for non-empty list.

**push\_front(int key, double value)**

* **Purpose:** Inserts a node at the beginning of the list.
* **Assumptions:** Valid inputs and available memory.
* **Inputs:** Integer key and double value.
* **Outputs:** None.
* **State Changes:** head is updated to the new node.
* **Cases and Expected Behavior:**
  + On empty list, node becomes the head.
  + On non-empty list, new node links to old head.

**push\_back(int key, double value)**

* **Purpose:** Adds a new node containing the specified key and value to the end of the linked list.
* **Assumptions:** The linked list has been initialized and there is sufficient memory to allocate a new node.
* **Inputs:** An integer key and a double value to be inserted at the end of the list.
* **Outputs:** None.
* **State Changes:** The list gains a new node at the end.
* **Cases and Expected Behavior:**
  + Empty List: New node becomes the head.
  + Non-empty List: Node is appended at the end.
  + Edge Case: No duplicates are checked; may insert identical key/value.

**insert(int key, double value, int index)**

* **Purpose:** Inserts a node at a specified index.
* **Assumptions:** Index is within bounds.
* **Inputs:** key, value, and index.
* **Outputs:** None.
* **State Changes:** New node is placed at index; pointers updated.
* **Cases and Expected Behavior:**
  + Index 0: Functions like push\_front.
  + Middle/End: Traverses and inserts.
  + Invalid Index: Operation is ignored.

**search(int key)**

* **Purpose:** Returns value associated with the given key.
* **Assumptions:** Key is assumed to be unique.
* **Inputs:** key (int).
* **Outputs:** Corresponding value (double), or -1.0 if not found.
* **State Changes:** None.
* **Cases and Expected Behavior:**
  + Key exists: Returns associated value.
  + Key not found: Returns sentinel (-1.0).

**at(int index)**

* **Purpose:** Returns pointer to the node at the given index.
* **Assumptions:** Index is 0-based and in range.
* **Inputs:** index (int).
* **Outputs:** Pointer to node or nullptr if index is invalid.
* **State Changes:** None.
* **Cases and Expected Behavior:**
  + Valid index: Returns correct node pointer.
  + Out-of-bounds: Returns nullptr.

**size()**

* **Purpose:** Returns number of nodes in the list.
* **Assumptions:** List can be traversed from head.
* **Inputs:** None.
* **Outputs:** Integer count of nodes.
* **State Changes:** None.
* **Cases and Expected Behavior:**
  + Empty list returns 0.
  + Otherwise returns accurate size.

**remove(int index)**

* **Purpose:** Removes node at the specified index.
* **Assumptions:** Index is within valid range.
* **Inputs:** index (int).
* **Outputs:** Boolean indicating success.
* **State Changes:** List length decreases by one.
* **Cases and Expected Behavior:**
  + Index 0: Removes head.
  + Middle/End: Adjusts pointers.
  + Invalid index: Returns false.

**remove\_key(int key)**

* **Purpose:** Removes first node matching the given key.
* **Assumptions:** Keys are unique.
* **Inputs:** key (int).
* **Outputs:** Boolean indicating success.
* **State Changes:** Node is deleted if found.
* **Cases and Expected Behavior:**
  + Key at head: Removes head.
  + Key in middle/end: Removes matching node.
  + Key not found: Returns false.

**remove\_value(double value)**

* **Purpose:** Removes first node with the specified value.
* **Assumptions:** Values may repeat.
* **Inputs:** value (double).
* **Outputs:** Boolean indicating success.
* **State Changes:** Node is deleted if value matches.
* **Cases and Expected Behavior:**
  + First match is removed.
  + No match: Returns false.

**print()**

* **Purpose:** Displays the contents of the list.
* **Assumptions:** Valid output stream (e.g., std::cout).
* **Inputs:** None.
* **Outputs:** Text output in the form (key, value) -> ... -> NULL.
* **State Changes:** None.
* **Cases and Expected Behavior:**
  + Empty list: Prints NULL.
  + Non-empty list: All nodes in order.

**selection\_sort()**

* **Purpose:** Sorts the list by key using selection sort.
* **Assumptions:** All keys are valid and comparable.
* **Inputs:** None.
* **Outputs:** None.
* **State Changes:** Node values are swapped to achieve sorting.
* **Cases and Expected Behavior:**
  + Empty or one-node list: No effect.
  + Multiple nodes: Sorted in ascending order.

**bubble\_sort()**

* **Purpose:** Sorts the list by key using bubble sort algorithm.
* **Assumptions:** All keys are valid and comparable.
* **Inputs:** None.
* **Outputs:** None.
* **State Changes:** Node values are swapped.
* **Cases and Expected Behavior:**
  + Empty or one-node list: No effect.
  + Unsorted list: Sorted by key in ascending order.
  + Already sorted: Minimal changes.

**UML Diagrams**

|  |
| --- |
| Node |
| + key : int  + value : double  + next : Node\* |
| + Node() :  + Node(k : int, v : double) : |

|  |
| --- |
| LinkedList |
| - head : Node\* |
| + LinkedList() :  + ~LinkedList() :  + get\_head() : Node\*  + push\_front(key : int, value : double) : void  + push\_back(key : int, value : double) : void  + insert(key : int, value : double, index : int) : void  + search(key : int) : double  + at(index : int) : Node\*  + size() : int  + remove(index : int) : bool  + remove\_key(key : int) : bool  + remove\_value(value : double) : bool  + print() : void  + selection\_sort() : void  + bubble\_sort() : void |