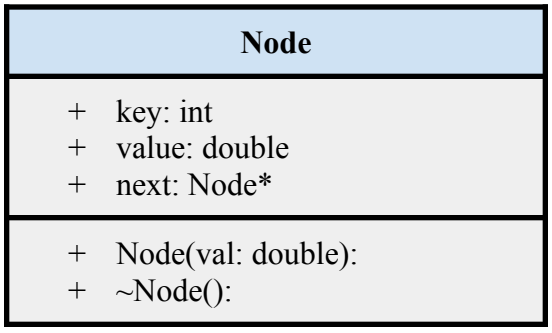
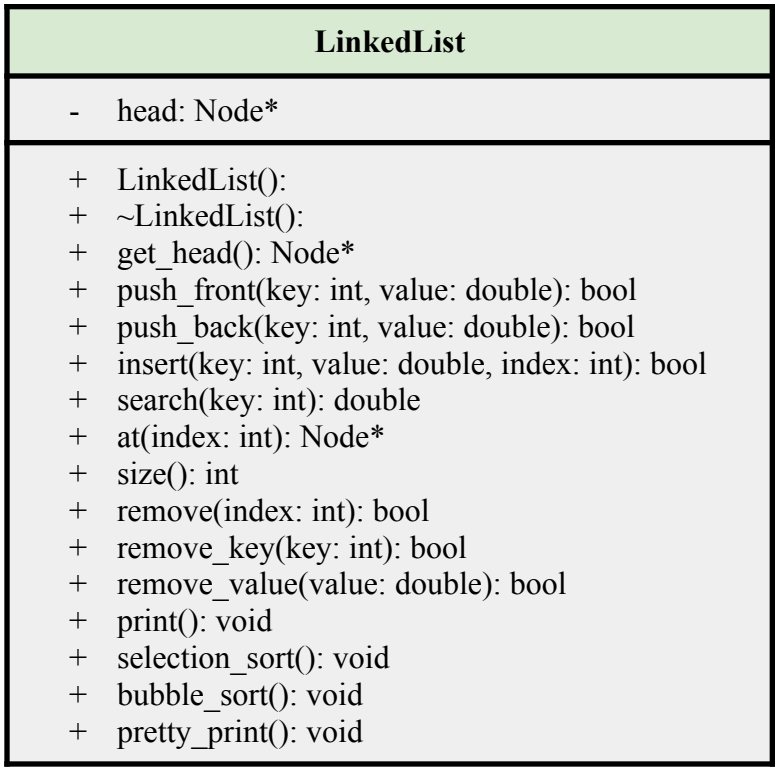
Gavin M, Aidan A, Darrell C, Megan G

CPSC122-02

4/13/2025

UML Diagrams





Specifications

---

bool remove(int index)

- **Purpose**: Remove a node from the linked list
- **Assumptions**: there is a linkedlist object
- **Inputs**: an integer index
- **Output**s: output bool for is successful or not
- **State Changes**: one node is removed from the list
- **Cases and Expected Behavior**: if the index is not in the list it's false. If indexes are equal to 0. Then the head will be removed

bool remove_key(int key)

- **Purpose**: Remove a node from the linked list based on the key
- **Assumptions**: there is a linkedlist object and the key exists in the linked list
- **Inputs**: an integer key
- **Output**s: output bool for is successful or not
- **State Changes**: one node is removed from the list
- **Cases and Expected Behavior**: If the index is not in the list it's false. If indexes are equal to 0. Then the head will be removed

bool push_back(int key, double value)

- **Purpose**: Create a new node with the given value and add it to the end of the linked list.
- **Assumptions**: There is a linked list that had enough memory to create a node.

- **Inputs**: an integer key and a double value
- **Outputs**: there is no output because the function is void
- **State Changes**: The new node should be added to the end of the linked lis
- **Cases and Expected Behavior**: If the linked list is empty, create a new dynamically allocated node, add the value to it, set the next pointer to nullptr and set head equal to its address. Otherwise, traverse to the end of the list, then create a new node, set the value to the given value and the next pointer to nullptr, and set the last node in the list's next pointer to the address of the new node.

bool push_front(int key, double value)

- **Purpose**: Create a new node with the given value and add it to the front of the linked list.
- **Assumptions**: There is a linked list that had enough memory to create a node.
- **Inputs**: an integer key and a double value
- **Outputs**: there is no output because the function is void
- **State Changes**: there will be a new node with the given value at the beginning of the list
- **Cases and Expected Behavior**: If the list is empty, there will be a single node with the next pointer set to nullptr. Otherwise, the list will be shifted back and the given value will be in the first node.

bool insert(int key, double value, int index);

- **Purpose:** Creates a new node with the given value and adds it to the linked list at the given index, moving nodes at or after that index back one

- **Assumptions:** We have a linked list and the value is appropriate for the list
- **Inputs:** An integer key, a double value, and the index value that we want to put the value
- **Outputs:** A boolean value indicating that the new node was inserted successfully
- **State Changes:** A new node is added to the linked list at the given index, moving all values at or after the given index back one
- **Cases and Expected Behavior:** If the index is greater than the size of the list or less than 0, return false, if the index is 0, the new node is added to the beginning of the list, otherwise, the function iterates through the list to the given index and inserts the new node, moving all nodes after it back by one.

bool remove_value(double value)

- **Purpose:** Removes the first instance of a value from a linked list
- **Assumptions:** There is a linked list
- **Inputs:** The value to be removed from the list
- **Outputs:** A boolean value indicating that the value was removed successfully
- **State Changes:** The first node containing the value will be removed from the list
- **Cases and Expected Behavior:** If the list does not contain the value to be removed, the function will return false, otherwise the function will iterate through the list until it finds the first instance of the given value and removes it from the list, moving all nodes after it one forward.

void print()

- **Purpose:** Print all values of a linked list
- **Assumptions:** There is a linked list
- **Inputs:** None
- **Outputs:** None
- **State Changes:** None
- **Cases and Expected Behavior:** If the list is empty, nothing is output to the console, otherwise, every key value pair will be output to the console

double search(int key)

- **Purpose:** Returns the value of the node based on a specified key
- **Assumptions:** There is a linked list object, and the inputted key is a valid key that exists within a node of that linked list object; there should not be more than one node with the same key.
- **Inputs:** An integer representing a key
- **Outputs:** The value being stored in the node with the inputted key value
- **State Changes:** None
- **Cases and Expected Behavior:** If the inputted key does not exist in the linked list object throw an error. If the inputted key does exist in the linked list object, return the value that is stored in the node that has that key. In the case that there are two nodes with the same key, return the value of the first node with that key.

Node* at(int index)

- **Purpose:** Returns the address of the node at the specified index

- **Assumptions:** There is a linked list object, and the inputted index is a valid index of that linked list object
- **Inputs:** An integer representing an index
- **Outputs:** The address in memory of the node at the inputted index
- **State Changes:** None
- **Cases and Expected Behavior:** If the inputted index does not exist in the linked list object we are trying to get the address from, return nullptr as a dummy value. If the inputted index does exist in the linked list object, return the address of the node at that index

int size()

- **Purpose:** Gives the number of items in the linked list
- **Assumptions:** There is a linked list object to check the size of
- **Inputs:** None
- **Outputs:** An integer value that describes the number of items in the list
- **State Changes:** None
- **Cases and Expected Behavior:** If the list is empty, the resultant integer should be 0; else if there are items in the list, this function will iterate through the items and count them, ultimately returning the total number of items.

void selection_sort()

- **Purpose:** Uses selection sort to sort the LinkedList in ascending order
- **Assumptions:** There is a LinkedList to sort with values that are able to be sorted

- **Inputs:** None
- **Outputs:** None
- **State Changes:** The order of the list is sorted in ascending order
- **Cases and Expected Behavior:** If the list is empty, nothing happens; else if there are items in the list, they will be sorted into ascending order

void bubble_sort()

- **Purpose:** sorts a linkedlist object
- **Assumptions:** There is a linked list object to sort
- **Inputs:** None
- **Outputs:** None
- **State Changes:** LinkedList will be sorted
- **Cases and Expected Behavior:** If the list is empty, if there are items in the linked list they are sorted decreasing to increasing order.