class Node

+ key : int
+ value : double
+ next : Node*

+ Node( )
+ Node(int k, double v)

class LinkedList

- head : Node*

+ LinkedList( )
+ ~LinkedList( )
+ get_head( ) const : Node*
+ push_front(int key, double value) : void
+ push_back(int key, double value) : void
+ insert(int key, double value, int index) : void
+ search(int key) : double
+ at(int index) : Node*
+ size( ) : int
+ remove(int index) : bool
+ remove_key(int key) : bool
+ remove_value(double value) : bool
+ print( ) : void
+ selection_sort( ) : void
+ bubble_sort( ) : void
+ print_stocks( ) : void
+ is_prime(int num) : bool
+ to_string( ) : string
+ stocks_string( ) : string

Global Data Members

+ problem_one( ) : void
+ problem_two( ) : void
+ problem_three( ) : void

LinkedList Specifications

LinkedList( )

- Purpose: default constructor, set head = nullptr
- Assumptions: class exists

- Inputs: none
- Outputs: none
- State changes: head = nullptr
- Cases & expected behavior: expected to set head to nullptr

~LinkedList( )
- Purpose: delete all nodes in LinkedList
- Assumptions: class exists to deconstruct
- Inputs: none
- Outputs: none
- State changes: linked list gets deleted
- Cases & expected behavior: expected to delete linked list, could not be a list

Node* get_head const ( )
- Purpose: return pointer to head of linked list
- Assumptions: head exists, class exists
- Inputs: none
- Outputs: pointer to head of linked list
- State changes: none
- Cases & expected behavior: expected to return pointer to head of linked list, could not be a list

void push_front(int key, double value)
- Purpose: add node with key and value to front of linked list
- Assumptions: there is a linked list
- Inputs: int key, double value
- Outputs: none
- State changes: node added to front of linked list
- Cases & expected behavior: linked list can be empty or already full, will either be first node or shift other nodes

void push_back(int key, double value)
- Purpose: add node with key and value to back of linked list
- Assumptions: there's a linked list
- Inputs: int key, double value
- Outputs: none
- State changes: node added to back of linked list
- Cases & expected behavior: linked list can be empty or already have items, expected to add node regardless

void insert(int key, double value, int index)
- Purpose: insert node with key, value at index location
- Assumptions: all parameters are provided, linkedlist exists
- Inputs: int key, double value, int index
- Outputs: none
- State changes: node inserted into linked list
- Cases & expected behavior: adds node if the index given was in bounds

double search(int key)
- Purpose: return value of first node with given key

- Assumptions: key is in linked list
- Inputs: int key
- Outputs: value of associated double, or -1 if not found
- State changes: none
- Cases & expected behavior: key may not be found, returns -1 if so

Node* at(int index)

- Purpose: return pointer to node at specified index
- Assumptions: index is valid and linked list exists
- Inputs: int index
- Outputs: pointer to node at specified index
- State changes: none
- Cases & expected behavior: returns nullptr if index is out of bounds

int size()

- Purpose: returns size of linked list
- Assumptions: linked list exists
- Inputs: none
- Outputs: integer representing size of linked list
- State changes: none
- Cases & expected behavior: returns 0 if linked list is empty, otherwise returns count, which is incremented to count the nodes

bool remove(int index)

- Purpose: removes node at specified index, returns false if not found, or true if found and removed
- Assumptions: linked list exists
- Inputs: int index
- Outputs: bool representing whether index was removed
- State changes: node removed at specified index (if it's in bounds)
- Cases & expected behavior: returns false if linked list is empty or index is out of range, otherwise returns true

bool remove_key(int key)

- Purpose: removes first node with specified key
- Assumptions: linked list exists
- Inputs: int key
- Outputs: boolean indicating whether node was removed
- State changes: first node with specified key is removed (if one exists)
- Cases & expected behavior: returns false if linked list is empty or key doesn't exist, otherwise returns true

bool remove_value(double value)

- Purpose: removes first node with specified value
- Assumptions: linked list exists
- Inputs: double value
- Outputs: boolean indicating whether first node with specified value was removed
- State changes: first node with specified value is removed (if one exists)

- Cases & expected behavior: if list empty or value not in list, returns false, otherwise returns true

void print( )
- Purpose: print key and value for each node in the linked list
- Assumptions: linked list exists
- Inputs: none
- Outputs: none
- State changes: none
- Cases & expected behavior: does nothing if list is empty

void selection_sort( )
- Purpose: sort linked list nodes based on key values (using selection sort)
- Assumptions: list has multiple nodes
- Inputs: none
- Outputs: none
- State changes: linked list sorted in ascending order (if already exists unsorted)
- Cases & expected behavior: does nothing if list is empty or already sorted, otherwise sorts it ascending

void bubble_sort( )
- Purpose: sort linked list nodes based on key values (using bubble sort)
- Assumptions: list has multiple nodes
- Inputs: none
- Outputs: none
- State changes: linked list sorted in ascending order (if already exists unsorted)
- Cases & expected behavior: does nothing if list is empty or already sorted

void print_stocks( )
- Purpose: prints key and value for each node in linked list, nicely formatted
- Assumptions: linked list with stock prices and rankings is created
- Inputs: none
- Outputs: none
- State changes: none
- Cases & expected behavior: prints top line if list is empty

bool is_prime(int num)
- Purpose: returns true if num is prime, otherwise false
- Assumptions: num is an integer
- Inputs: int num
- Outputs: boolean that's true if num is prime, otherwise false
- State changes: none
- Cases & expected behavior: returns false if num is zero or negative

string to_string( )
- Purpose: convert linked list keys and values to a string
- Assumptions: linked list has at least one node
- Inputs: none
- Outputs: string with all linked list values
- State changes: none

- Cases & expected behavior: returns “” if linked list is empty

string stocks_string( )

- Purpose: convert stocks linked list to string
- Assumptions: linked list has at least one node
- Inputs: none
- Outputs: none
- State changes: none
- Cases & expected behavior: returns header string if list empty