# Stations

## Searching

Initialize a node pointer, current = head.

Do following while current is not NULL

 If the current value (i.e., current->key) is equal to the key being searched return true.

Otherwise, move to the next node (current = current->next).

If the key is not found, return false

BEGIN

Declare search

Declare found=false

Declare Station\*node

\*node=head

While(node!=null)

If(search==node->station\_name)

Found=true

Display node-> station\_ID

Display node-> station\_name

Display node->

Display node->

Display node->

Display node->

ENDIF

Node=node->next

ENDWHILE

END

# Transactions

## Searching

FUNCTION search\_passenger(arr, length, passenger)

BEGIN

IF length = 0 THEN

RETURN NULL

ENDIF

DECLARE mid = length/2

IF passenger = arr[mid].name THEN

RETURN arr[mid]

ENDIF

IF passenger < arr[mid].name THEN

RETURN search\_passenger(arr, mid, passenger)

ENDIF

RETURN search\_passenger(arr[mid], mid, passenger)

## Sorting

Based on https://www.geeksforgeeks.org/merge-sort-for-linked-list/

FUNCTION sort\_transactions(transactions)

BEGIN

DECLARE head = transactions

IF head = NULL OR head.next = NULL THEN

END FUNCTION

ENDIF

DECLARE mid = head

DECLARE end = head.next

DOWHILE end != NULL

end = end.next

IF end != NULL THEN

mid = mid.next

end = end.next

ENDIF

ENDDO

mid.next = NULL

sort\_transactions(head)

sort\_transactions(mid)

transactions = merge(head, mid)

END

FUNCTION merge(a, b)

BEGIN

DECLARE result = NULL

IF a = NULL THEN

RETURN b

ELSE

IF b = NULL THEN

RETURN a

ENDIF

ENDIF

IF a.date <= b.date THEN

result = a

result.next = merge(a.next, b)

ELSE

result = b

result.next = merge(a, b.next)

ENDIF

RETURN result

END