A2 List ADT - Algorithm Designs

(*Note: implementation may differ, but general concepts from here still apply.*)

Insertion Methods Pseudocode

(Members of **List** class)

```
// helper function (will be private member)
// assumes pointer argument points to an item in the list
void insert(const listDataType& entry, ListItem* listPtr)    // general insert
        INIT newItemPtr to NEW ListItem
        SET newItemPtr item data to entry

        IF list is empty
                SET headPtr to newItemPtr
                SET tailPtr to newItemPtr
        ELSE IF listPtr EQUAL TO headPtr
                SET newItemPtr's next pointer to headPtr
                SET headPtr item prev field to newItemPtr
                SET headPtr to newItemPtr
        ELSE IF listPtr EQUAL TO tailPtr
                SET newItemPtr's prev pointer to tailPtr
                SET tailPtr's item next field to newItemPtr
                SET tailPtr to newItemPtr
        ELSE (inserts item AFTER listPtr)
                SET newItemPtr's prev pointer to listPtr
                SET newItemPtr's next pointer to listPtr's next field

                SET listPtr's item next field to newItemPtr
                SET item after listPtr's prev field to newItemPtr
        ENDIF

        INCREMENT size of list
        SET listPtr to nullptr

        (Note: allocated memory is released by destructor when it goes out of scope here)
END insert method

void insertToTail(const listDataType& entry)
        CALL insert(entry, tailPtr)
END insert method

void insertToHead(const listDataType& entry)
        CALL insert(entry, headPtr)
```

END insert method

void **insertToMid**(const listDataType& entry)
    INIT listPtr to headPtr of list
    INIT item to 1

    LOOP while count IS LESS THAN size of list divided by 2 (*find middle*)
        SET listPtr to listPtr's next field (*traverse list*)
        INCR count
    ENDLOOP

    CALL **insert**(entry, listPtr)

    SET listPtr to nullptr // *avoid dangling pointers!*
END insert method

Delete Item Pseudocode

// deletes item at head of list
void **removeFromHead**()
    IF list is NOT empty
        INIT temp pointer (to delete item at head)
        SET temp to headPtr

        SET headPtr to item AFTER temp

        IF headPtr NOT nullptr (*means list had one item*)
            SET headPtr's prev field to nullptr
        ELSE
            SET tailPtr to nullptr
        ENDIF

        DELETE ListItem pointed by temp (*free memory*)
        DECR size of list
    ELSE
        PRINT message "List is empty"
    ENDIF
END removeFromHead method

// deletes item at any location in list (helper function)
void **remove**(int key)
    INIT temp as pointer to ListItem

```
    SET temp to find(key)          // return nullptr if not found

    IF list is empty
        PRINT "List is empty"
    ELSE IF temp EQUAL TO nullPtr
        PRINT "Invalid key"
    ELSE IF temp EQUAL TO headPtr              (removing from head)
        SET headPtr to temp's next field.     (CAN BE NULLPTR!)

        // if list after delete has more than one item, set headPtr to next item
        IF headPtr NOT EQUAL TO nullptr
            SET headPtr's prev to nullptr
        ELSE
            SET tailPtr to nullptr
        ENDIF
    ELSE IF temp EQUAL TO tailPtr              (removing from tail)
        SET tailPtr to temp's prev field

        // if list after delete has more than one item, set tailPtr to next item
        IF tailPtr NOT EQUAL TO nullptr
            SET tailPtr next to nullptr
        ELSE
            SET headPtr to nullptr
    ELSE                                       (removing from middle)
        SET previous item's next to temp's next field
        SET proceeding item's prev to temp's prev field
    ENDIF

    DELETE ListItem pointed by temp            (free memory)
    SET temp to nullptr
END remove method
```

Search Method Pseudocode

(Member of **List** class)

bool **search**(const listDataType& data)

```
    LOOP over each item in list
        IF current item's data IS EQUAL TO data
            RETURN true
        ENDIF
    ENDLOOP
```

```
        RETURN false
END search method
```

Sort Methods Pseudocode (uses selection sort)

(Member of **List** class)

```
void sortAsc()
        INIT minPtr to headPtr
        INIT compItemPtr to headPtr->getNext()        // comparison item
        INIT tempPtr to NEW ListItem

        LOOP while compItem NOT EQUAL TO nullptr
                IF compItemPtr item data IS LESS THAN minPtr item's
                        SWAP compItemPtr's data with minPtr's item's data
                ENDIF
        ENDLOOP

END sort method
```