3. List

Problem 3.1

1. Write the method `public <T> static void clear(ArrayList<T> l)` which removes all the elements of *l*. What would you need to change if the list `l` were of type `LinkedList` ?

2. Write method `insertAll` as user of ADT List that takes two lists `l1` , `l2` and index `i` and insert all elements in `l2` in `l1` after position `i` . The list `l2` must not be changed. The first element has position 0, and assume that `i` is a valid position.

   ■ **Example 3.1** If $l1 : A, B, C, D$, and $l2 : X, Z$, then after calling `insertList(l1, l2, 1)` , then $l1 : A, B, X, Z, C, D$. ■

   Method signature `public <T> void insertAll(List<T> l1, List<T> l2, int i)` .

3. Write the method: `public static <T> void commonE(List<T> l1, List<T> l2, List<T> cl)` , user of the ADT List, which inserts the common elements between list `l1` and list `l2` in the list cl. Assume that the elements in `l1` and `l2` are unique and the List `cl` is initially empty.

   ■ **Example 3.2** If $l1 : A, B, C, F, M, D$, and $l2 : R, M, W, F$, calling `commonE(l1, l2, cl)` results in $cl : F, M$ ■

4. Write the method `moveToEnd` , user of the ADT `List` . The method takes a list `l` and an index `i` . It will move the element at the `i` -th position to the end of the list. You can assume `i` to be within the list, and that the first element has the position 0. Do not use any auxiliary data structures. The method signature is: `public static <T> void moveToEnd(List<T> l, int i)` .

   ■ **Example 3.3** If $l : a \rightarrow c \rightarrow d \rightarrow b \rightarrow r \rightarrow x$, then after calling `moveToEnd(l, 2)` , `l` will be: $a \rightarrow c \rightarrow b \rightarrow r \rightarrow x \rightarrow d$. ■

5. ★ Write a static method `public static <T> T mfe(List<T> l)` (user of ADT) that takes as input a non-empty list `l` and returns the most frequent element in the list `l` . If two or more elements appear the same number of times, then the earliest one to appear in the list

should be the most frequent one.

■ **Example 3.4** Assuming $l : 1, 2, 3, 4, 2, 5, 3$. Calling `mfe(l)` will return: 2.   ■

Problem 3.2

1. Write the method `filter`, member of the class `LinkedList` that takes as parameter an object that implements the interface `Condition` below. The method removes all the elements of the list for which the method `test` returns **false**. The method signature is `void filter(Condition<T> cnd)`.

    ```
    public interface Condition <T> {
        boolean test (T data);
    }
    ```

2. Write the method `traverse`, member of the class `LinkedList` that takes as parameter an object that implements the interface `Processor` below. The methods traverses the list and call the method `process` on all elements of the list.

    ```
    public interface Processor <T> {
        void process (T data);
    }
    ```

3. Write the method `removeBetween`, member of the class `LinkedList`. The method takes two elements `e1` and `e2`, and removes all the elements between the two elements (`e1` and `e2` not included). If `e1` or `e2` or both do not exist, no element will be removed. You can assume the elements of the list to be unique, and that `e1` $\neq$ `e2`. Do not call any methods and do not use any auxiliary data structures. The method signature is: `public void removeBetween(T e1, T e2)`.

    ■ **Example 3.5** If the list: $a \rightarrow c \rightarrow d \rightarrow b \rightarrow r \rightarrow x$, then after calling `removeBetween(''c'', ''r'')`, the list becomes: $a \rightarrow c \rightarrow r \rightarrow x$.   ■

4. As a member of the class `LinkedList`, write the method `public void insertBefore(T e, int i)` that inserts the element e before the `ith` element. The numbering starts from 0. Assume that `i` is a valid index. Do not call any methods of the class `LinkedList`. Do not use any auxiliary data structures.

    ■ **Example 3.6** If $l : A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$, then calling `l.insertBefore('N', 4)` changes the list to $l : A \rightarrow B \rightarrow C \rightarrow D \rightarrow N \rightarrow E$. Calling `l.sublist('N', 0)`, changes the list to $l : N \rightarrow A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$.   ■

5. Implement the following methods in the class `LinkedList`:
    (a) **Procedure** insertBeforeCurrent(T e). **Requires**: The list l should not be full. **Results**: The new element e is inserted before the current and the new element is made the current.
    (b) **Procedure** removeIth(int i). **Requires**: The list l should not be empty. **Results**: The element e at position i is removed from the list (numbering starts with 0), If the resulting list is empty current is set to NULL. If successor of the deleted element exists it is made the new current element otherwise first element is made the new current element.

6. Write the method `removeOddElems`, member of the class `LinkedList`, that removes all the elements having an odd position (the position of the first element is 0). The method signature is: `public void removeOddElems()`. **Do not call any methods and do not use any auxiliary data structure**.

■ **Example 3.7** If $l : A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$, then `l.oddElems()` returns: $A \rightarrow C \rightarrow E$. ■

7. ★ Write the method `removeFirst`, member of class `LinkedList`, that removes the first occurrence of every element that **appears more than once** in the list. Do not use any auxiliary data structures and do not call any methods. The method signature is **public void** `removeFirst()`.

■ **Example 3.8** If the list contains: $B \rightarrow A \rightarrow A \rightarrow R \rightarrow C \rightarrow A \rightarrow C \rightarrow R$, then after calling `removeFirst`, its content becomes $B \rightarrow A \rightarrow A \rightarrow C \rightarrow R$. ■

Problem 3.3

1. Consider the function `f` below, member of `DoubleLinkedList`:

```
public void f(int n) {
    Node<T> p = head, q;
    for(int i = 0; i < n; i++)
        if(p.next != null)
            p = p.next;

    if(p != null && p.next != null){
        q = p;
        while(q.next != null)
            q = q.next;
        q.previous.next = null;
        q.previous = null;
        q.next = p;
        p.previous = q;
        head = q;
    }
}
```

Give the content of the list after each of the following cases:
   (a) The list `l` : A, B, C, D, E, after calling `l.f(1)`.
   (b) The list `l` : A, B, C, D, E, after calling `l.f(0)`.
   (c) The list `l` : A, B, C, D, E, after calling `l.f(2)`.
   (d) The list `l` : A, B, C, D, E, after calling `l.f(5)`.
2. Consider the function `f` below, member of `ArrayList`:

```
void f(int i, int j) {
    for (int k = i; k <= j; k++) {
        T e1 = data[k];
        T e2 = data[k + j - i + 1];
        boolean flag = false;
        for(int l = 0; l < i; l++) {
            if (e1.equals(data[l])) {
                flag = true;
                break;
            }
        }
        if (!flag) {
            data[k] = e2;
            data[k + j - i + 1] = e1;
        } else {
            flag = false;
            for(int l = j + 1; j < size; l++) {
                if (e1.equals(data[l])) {
                    flag = true;
                    break;
```

```
            }
        }
        if (!flag) {
            data[k] = e2;
            data[k + j - i + 1] = e1;
        }
    }
  }
}
```

Give the content of the list after each of the following cases:
  (a) The list `l` : A, F, C, E, B, D, F, D, after calling `l.f(2, 3)` .
  (b) The list `l` : D, B, E, F, B, B, B, E, after calling `l.f(3, 4)` .
  (c) The list `l` : D, A, F, B, C, A, B, A, after calling `l.f(4, 4)` .
  (d) The list `l` : A, A, E, E, A, E, D, E, after calling `l.f(3, 5)` .
  (e) The list `l` : C, F, B, E, F, E, B, D, after calling `l.f(0, 2)` .
3. Consider the function `f` below, member of `LinkedList` :

```
void f(int k) {
    Node<T> p = current;
    Node<T> q = current.next;
    for(int i = 0; i < k; i++) {
        Node<T> tmp = q.next;
        q.next = p;
        p = q;
        q = tmp;
    }
    current.next = q;
    if (current == head) {
        head = p;
    } else {
        q = head;
        while (q.next != current)
            q = q.next;
        q.next = p;
    }
}
```

Give the content of the list after each of the following cases:
  (a) The list `l` : A, B, C, D, E, F, G, current on C, after calling `l.f(2)` .
  (b) The list `l` : A, B, C, D, E, F, G, current on A, after calling `l.f(3)` .
  (c) The list `l` : A, B, C, D, E, F, G, current on E, after calling `l.f(2)` .
  (d) The list `l` : A, B, C, D, E, F, G, current on D, after calling `l.f(0)` .
  (e) The list `l` : A, B, C, D, E, F, G, current on A, after calling `l.f(6)` .
4. Consider the function `f` below, member of `DoubleLinkedList` :

```
void f(int i, int j) {
    Node<T> q = current;
    while (q.next != null)
        q = q.next;
    Node<T> tail = q;
    Node<T> p = head;
    for (int k = 0; k < i; k++)
        p = p.next;
    for (int k = 0; k < j; k++)
        q = q.previous;
    while (p != q && q.next != p) {
        T e1 = p.data;
        T e2 = q.data;
        boolean flag1 = false;
```

```
        Node<T> t = head;
        for (int k = 0; k < i; k++) {
            if (e2.equals(t.data)) {
                falg1 = true;
                break;
            }
            t = t.next;
        }
        boolean flag2 = false;
        Node<T> t = tail;
        for (int k = 0; k < j; k++) {
            if (e1.equals(t.data)) {
                falg2 = true;
                break;
            }
            t = t.previous;
        }
        if (flag1 == flag2) {
            p.data = e2;
            q.data = e1;
        }
        p = p.next;
        q = q.previous;
    }
}
```

Give the content of the list after each of the following cases:

(a) The list `l` : A, F, C, E, B, D, F, D, after calling `l.f(2, 1)` .

(b) The list `l` : D, B, E, F, B, B, B, E, after calling `l.f(3, 0)` .

(c) The list `l` : D, A, F, B, C, A, B, A, after calling `l.f(1, 2)` .

(d) The list `l` : A, A, E, E, A, E, D, E, after calling `l.f(0, 3)` .

(e) The list `l` : C, F, B, E, F, E, B, D, after calling `l.f(2, 2)` .

5. Consider the function `f` below, member of `LinkedList` :

```
void f(int i, int j, T e) {
    Node<T> p = head;
    for (int k = 0; k < i; k++)
        p = p.next;
    Node<T> q = p;
    for (int k = 0; k < j; k++)
        q = q.next;
    Node<T> r = head;
    for (int k = 0; k < i - 1; k++)
        r = r.next;
    if (r == p)
        r = null;
    Node<T> t = p;
    boolean falg = false;
    for (int k = 0; k < j; k++) {
        if (e.equals(t.data)) {
            flag = true;
            break;
        }
        t = t.next;
    }
    if (flag) {
        if (r == null)
            head = q.next;
        else
            r.next = q.next;
```

```
          current = head;
      }
}
```

Give the content of the list after each of the following cases:

(a) The list `l` : A, F, C, E, B, D, F, D, after calling `l.f(2, 3, 'E')`.
(b) The list `l` : D, B, E, F, B, B, B, E, after calling `l.f(0, 4, 'C')`.
(c) The list `l` : D, A, F, B, C, A, B, A, after calling `l.f(3, 4, 'C')`.
(d) The list `l` : A, A, E, E, A, E, D, E, after calling `l.f(0, 7, 'E')`.
(e) The list `l` : C, F, B, E, F, E, B, D, after calling `l.f(0, 1, 'C')`.

Problem 3.4

1. Write the method `removeEvenElems`, member of the class `ArrayList`, that removes all the elements having an even position (the position of the first element is 0). The method must run in $O(n)$. The method signature is: `public void removeEvenElems ()`. **Do not call any methods and do not use any auxiliary data structure**.

   ■ **Example 3.9** If l : A, B, C, D, E, then after calling the method `l.removeEvenElems()` l becomes: B, D.   ■

2. Write the method `duplicate`, member of the class `ArrayList`, that duplicates each element of the list putting the duplicate right after the original. The method must run in $O(n)$. The method signature is: `public void duplicate()`. **Do not call any methods and do not use any auxiliary data structure.**

   ■ **Example 3.10** If l : A, A, B, C, B, then after the call `l.duplicate()`, l becomes: A, A, A, A, B, B, C, C, B, B.   ■

Problem 3.5

1. Write the method `checkListEndsSymmetry` that receives a double linked list and an integer number `k`. The method checks if the double linked list has identical `k` elements going forward from the first element and backwards from the last one. The method returns **true** if they are identical, and **false** otherwise. The method signature is:
   `public <T> boolean checkListEndsSymmetry(DoubleLinkedList<T> dl, int k)`

   ■ **Example 3.11** If $dl = A \leftrightarrow B \leftrightarrow C \leftrightarrow D \leftrightarrow B \leftrightarrow A$ and $k = 2$, then the method should return **true**. If $k = 3$, it should return **false**, since C does not equal D.

                                                                                        ■

2. ★ Write the method `bubbleSort` that sorts a double linked list of integers given as input using bubble sort. The method signature is: `public void bubbleSort(DoubleLinkedList<Integer> l)`.

Problem 3.6

A double linked list with sentinel nodes has special header and trailer nodes that do not store data (see Figure 3.1). Therefore, all nodes that store data have previous and next nodes, which eliminates special cases from insert and remove.
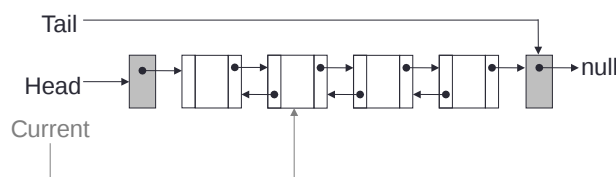
Figure 3.1: A double linked list with sentinel nodes

Write down the code for this implementation (class `DLLSentinel`).

Problem 3.7

One of the limitations of the array implementation of the ADT List seen in class is the size limit. It is, however, possible to solve this problem through a dynamic reallocation of the array as follows.

- The size of the array is initially set to a small value (say 1).
- If the capacity of the array becomes insufficient, a new array is allocated having double the size of the current one. The data is then transferred from the old array to the new one, and the old array is discarded.
- If percentage of used space drops below a given value (for example 40%), a new array is allocated having half the size of the current one. The data is then transferred from the old array to the new one, and the old array is discarded.

Complete the implementation of the class `DArrayList` that uses this technique: Write the methods: `full`, `insert` and `remove`.

```java
public class DArrayList<T> implements List<T> {
    private T[] data;
    private int current, size, maxSize;
    private static final double minRatio = 0.4;
    public DArrayList() {
        data = (T[]) new Object[1];
        maxSize = 1;
        current = -1;
        size = 0;
    }
}
```

Problem 3.8

A circular list is a list with no first or last element and where the current advances in a circular way through the data.

1. Give a clear specification of the ADT `CircularList` (use the same format seen in lecture).
2. Write down the interface `CircularList`.
3. Write the method `public void print(CircularList<String> l)`, which prints the content of `l` starting from the current element. At the end of the method, current must return to its initial position.
4. Give a linked implementation of the interface `CircularList` (class `LinkedCircularList`).

Problem 3.9

In a library management application, information about books and authors are stored in the two classes `Book` and `Author` shown below. Notice that a book may have multiple authors, and an author may participate in writing several books.

```java
public class Book {
```

```
    public String isbn;
    public String title;
    public List<Author> authors; // Authors of this book
    public Book(String isbn, String title, List<Author> authors) {
        this.isbn= isbn;
        this.title= title;
        this.authors= authors;
    }
}
```

```
public class Author {
    public String firstName;
    public String lastName;
    public Author(String firstName, String lastName) {
        this.firstName= firstName;
        this.lastName= lastName;
    }
}
```

1. Write the method `authorsOf` that takes as input a list of books and a book's ISBN and returns a list containing all the authors of this book if it exists in the list, empty list otherwise. The method signature is: `List<Author> authorsOf(List<Book> books, String isbn)`.
2. Write the method `booksBy` that takes as input a list of books and an author's last name and returns a list of all books written by this author. The method signature is: `List<Book> booksBy(List<Book> books, String lastName)`.

Problem 3.10

A polynomial can be represented as a list, where each node contains two numbers: an integer that indicates the degree of the variable x and a real number that gives the corresponding coefficient. Degrees of x that have null coefficients (i.e. 0) are not included in the list (therefore, the null polynomial is represented by the empty list.) Furthermore, the degrees of x appear in strictly increasing order.

■ **Example 3.12** The polynomial $2x^2 - x + 3$ is represented as:

$$(0, 3.0) \to (1, -1.0) \to (2, 2.0),$$

whereas $x^4 + 3x^2$ is represented by:

$$(2, 3.0) \to (4, 1.0).$$

■

The data contained in the list is represented by the class Monomial:

```
public class Monomial {
    public int deg; // The degree of x
    public double coef; // The coefficient

    public Monomial(int d, double c){
        deg= d;
        coef= c;
    }
}
```

1. Write the static method `sumPol`, member of the class `PolynomialOp`, that takes as input two polynomials (of type `List<Monomial>`) and returns a new polynomial which is the sum of the two arguments. The input polynomials must not be modified.

2. ⭐ Write the static method `prodPol`, member of the class `PolynomialOp`, that takes as input two polynomials (of type `List<Monomial>`) and returns a new polynomial which is the product of the two arguments. The input polynomials must not be modified.

The methods signatures and its class are as follows:

```
public class PolynomialOp {
    public static List<Monomial> sumPol(List<
        Monomial> p, List<
        Monomial> q){
    }
    public static List<Monomial> prodPol(List<Monomial> p, List<
        Monomial> q){
    }
}
```

Problem 3.11

We want to implement the data structure `OrdList`, which consists of a list where the elements are ordered according to their *keys*. The key of an element is an integer that is given at the time of insertion.

■ **Example 3.13** This is an example of an ordered list where the data is of type String (the numbers represent the keys, notice the order):

$$(1,"A") \to (3,"H") \to (8,"C") \to (8,"B") \to (10,"R")$$

■

The specification of this data structure is as follows:
- Domain:
    - The elements of the list are of type `OrdListElem<T>` defined as follows:

    ```
    public class OrdListElem <T> {
        public int key;
        public T data;
        public OrdListElem(int k, T val) {
            key= k;
            data= val;
        }
    }
    ```

    - Structure: Linear with elements ordered in **increasing order** of the keys.
- Operations (all operations are done on an ordered list `l`):
    - Procedure `empty`: Same as a List.
    - Procedure `full`: Same as a List.
    - Procedure `last`: Same as a List.
    - Procedure `findFirst`: Same as a List.
    - Procedure `findNext`: Same as a List.
    - Procedure `last`: Same as a List.
    - Procedure `update(val: T)`: Same as a List (only the data of current element is updated, not its key).
    - Procedure `retrieve (elem: OrdListElem<T>`):
        * Preconditions: `l` is not empty.
        * Results: the key and data of `elem` are set to the key and data of the current element.
        * Method signature: **public** `OrdListElem<T> retrieve()`.
    - Procedure `insert (elem: OrdListElem<T>)`:

           ∗ Preconditions: `l` is not full.
           ∗ Results: if `l` is empty, then `elem` becomes the first element of the list. If `l` is not empty, then `elem` is inserted at the position corresponding to its key (**the position of** `current` **is irrelevant**). If there are other elements with the same key as `elem` in `l`, then `elem` must be inserted after them. After insertion, the newly inserted element becomes the current element.
           ∗ Method signature: **public void** insert (OrdListElem<T> elem) .
      – Procedure `remove` : Same as a List.

Write a linked implementation of this data structure (the class name is `LinkedOrdList<T>` , and its constructor does not take any parameters. The class representing the nodes is named `OLNode<T>`).

```
public class OLNode <T> {
    ...
    public OLNode (...) {...}
}
```



```
public class LinkedOrdList <T> {
    ...
    public LinkedOrdList () {...}
    public boolean empty () {...}
    public boolean full () {...}
    public boolean last () {...}
    public void findFirst () {...}
    public void findNext () {...}
    public void update (T val) {...}
    public OrdListElem <T> retrieve () {...}
    public void insert (OrdListElem <T> elem) {...}
    public void remove () {...}
}
```

Problem 3.12

A map containing *n* cities which are connected by roads is represented as a list of lists. The length of the list is *n*, and each element *i* in this list contains the list of the cities that are connected to city *i* (if a city *i* is connected to a city *j*, then *j* is also connected to *i*). The information about a road is contained an object of class *Edge* shown below.

```
public class Edge {
    public int i; // The starting node
    public int j; // The end node
    public int w; // The weight
    public Edge(int i, int j, int w) {
        this.i = i;
        this.j = j;
        this.w = w;
    }
}
```

■ **Example 3.14** The map in Figure 3.2 is represented as follows:

$$\square \quad \to (0,1,1) \to (0,2,2) \to (0,4,1)$$
$$\downarrow$$
$$\square \quad \to (1,0,1) \to (1,2,3)$$
$$\downarrow$$
$$\square \quad \to (2,0,2) \to (2,1,3) \to (2,3,2)$$
$$\downarrow$$
$$\square \quad \to (3,2,2) \to (3,4,1)$$
$$\downarrow$$
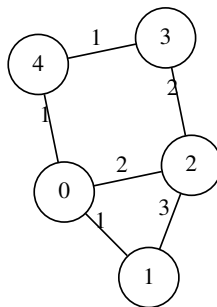$$\square \quad \to (4,0,1) \to (4,3,1)$$

Figure 3.2: A map

Consider the class `Map` below.

1. Write the method `nbLess(int d)` that returns the number of roads with length at most d (count only edges where $i < j$).
2. Write the method `getRoads(int d)` that returns all the edges that have exactly length d (return only edges where $i < j$).
3. The method `addRoad` adds a road between the two cities i and j. As a precondition, assume that: $i \neq j$, $0 \leq i, j < n$ and that there was previously no road between these two cities.
4. The method `validRoute` takes as input a list `route` of cities (integers) which represents a route (a sequence of cities). The method returns true if each city in the list `route` has a road to the following city. Assume that `route` is not empty and that each element in it is larger or equal 0 and smaller than n.

```java
public class Map {
    private int nbCities; // The number of cities.
    private List<List<Edge>> roads; // The roads
    ...
    public int nbLess(int d) {
    }
    public List<Edge> getRoads(int d) {
    }
    public void addRoad(int i, int j) {
    }
    public boolean validRoute(List<Integer> route) {
    }
}
```

Problem 3.13

★ ★   The goal of this problem to write a method that generates all the subsets of a given set of elements.

1. Write down all the subsets of the set $\{1,2,3\}$. What is the number of subsets of a set of n elements?

2. These are the subsets of the set $\{1,2\}$: $\{\emptyset, \{1\}, \{2\}, \{1,2\}\}$. Describe how you can get the subsets of $\{1,2,3\}$ from this set.

3. In general, suppose you have a set of n elements. If you are given the set containing all the subsets of $n-1$ elements how can you get the subsets of the whole set?

4. Write the method `public <T> List<List<T>> subsets(List<T> l)`, that returns all the subsets of the list `l` (call the **recursive** method `recSubsets`). Use the method `public List<T> copy()`, member of the interface *List* to make a copy of the list.