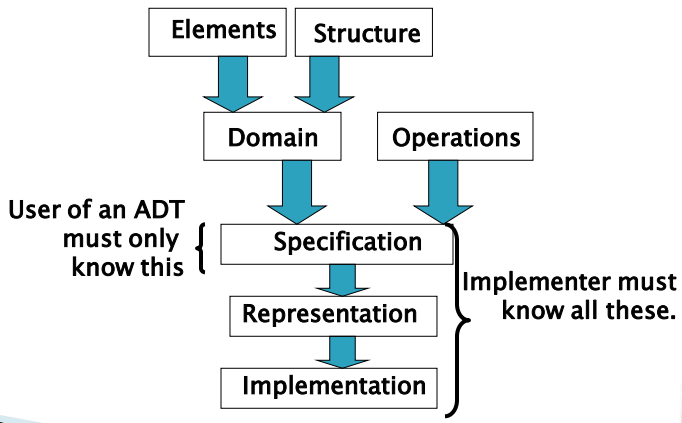
ADT List (LinkedList)

CS212 Data Structure
1st Semester 2011/2012
Lecture 4

ADT List



ADT List: Specification

Elements: The elements are of type <Type>. The **elements** are placed in **nodes** for linked list implementation.

```
public class Node<T> extends Object {
  public T data;
  public Node<T> next;
  public Node () {
     data = null;  next = null; }
  public Node (T val) {
     data = val;  next = null; }
}
```

Note: To implement encapsulation, we can have getter/setter of data and next here.

3

ADT List: Specification

Structure: the elements are linearly arranged, the first element is called head, there is a element called current, and there is a last element.

Domain: the number of elements in the list is bounded therefore the domain is finite. Type name of elements in the domain: List

4

ADT List: Specification

Operations: We assume all operations operate on a list L.

1. **Method** FindFirst ( )
   requires: list L is not empty. **input:** none
   results: first element set as the current element. **output:** none.
2. **Method** FindNext ( )
   requires: list L is not empty. Cur is not last. **input**: none
   results: element following the current element is made the current element.
   output: none.
3. **Method** Retrieve (Type e)
   requires: list L is not empty. **input**: none
   results: current element is copied into e. **output**: element e.

5

ADT List: Specification

Operations:

4. **Method** Update (Type e).
   requires: list L is not empty. **input**: e.
   results: the element e is copied into the current node.
   output: none.
5. **Method** Insert (Type e).
   requires: list L is not full. **input**: e.
   results: a new node containing element e is created and inserted after the current element in the list. The new element e is made the current element. If the list is empty e is also made the head element. **output**: none.

6

ADT List: Specification

Operations:

6. **Method** Remove ( )
   requires: list L is not empty. **input**: none
   results: the current element is removed from the list. 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. **output**: none.
7. **Method** Full (boolean flag)
   input: none. **returns**: if the number of elements in L has reached the maximum number allowed then flag is set to true otherwise false. **output**: flag.

7

ADT List: Specification

Operations:

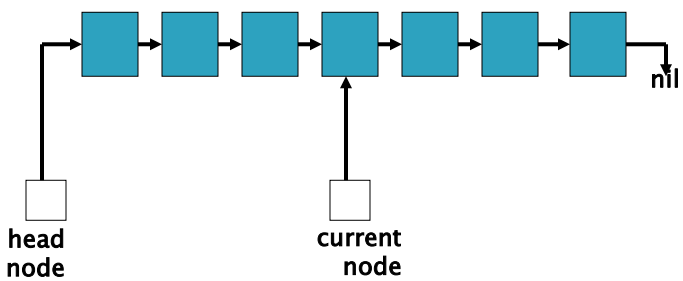
8. **Method** Empty (boolean flag).
   input: none. **results**: if the number of elements in L is zero, then flag is set to true otherwise false. **Output**: flag.
9. **Method** Last (boolean flag).
   input: none. **requires**: L is not empty. **Results**: if the last element is the current element then flag is set to true otherwise false. **Output**: flag

8

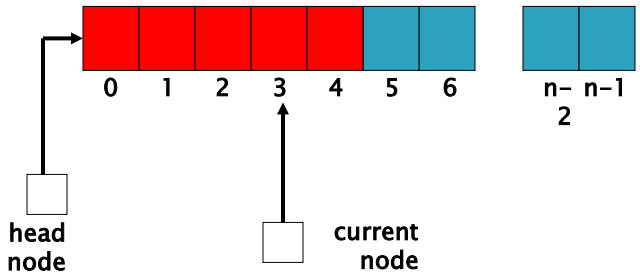
Implementation

- Programmers have to deal with the questions:
  - How to represent lists? Storage structure affects the efficiency of the operations.
  - How to implement the operations? Algorithm chosen must be efficient.
- Lists can represented as
  - Linked List
  - Array based List

Linked List



Array Based List



ADT List – Linked List

Representation:

```
public class LinkList<T> extends Object {
  private Node<T> head;
  private Node<T> current;
  public LinkList () {
     head = current = null; }
  public boolean empty () {
     return head == null; }
  public boolean last () {
     return current.next == null;}
```

ADT List: Implementation

```
public boolean full () {
   return false;}
public void findfirst () {
   current = head;}
public void findnext () {
   current = current.next;}
public T retrieve () {
   return current.data;}
public void update (T val) {
   current.data = val;}
```

ADT List: Implementation

```
public void insert (T val) {

Node<T> tmp;
      if (empty()){
          current = head = new Node<T> (val);
      }
      else {
   tmp = current.next;
   current.next = new Node<T> (val);
   current = current.next;
   current.next = tmp;}
}
```

ADT List: Implementation

```
public void remove () {
   if (current == head) {
       head = head.next; }
   else {
       Node<T> tmp = head;
       while (tmp.next != current) tmp = tmp.next;
       tmp.next = current.next; }
   if (current.next == null) {
       current = head; }
   else {
       current = current.next; }
   }
}
```

ToDo

- Read Text book 3.2, 6.2
- Add “int length()” method in the LinkList and ArrayList class, that should return the total number of elements in the class.
- Add “boolean findItem(T item)” that will return true if item is found otherwise false.
- Test These ADTs using a test Class.