Διάλεξη 07: Λίστες Ι – Υλοποίηση & Εφαρμογές

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

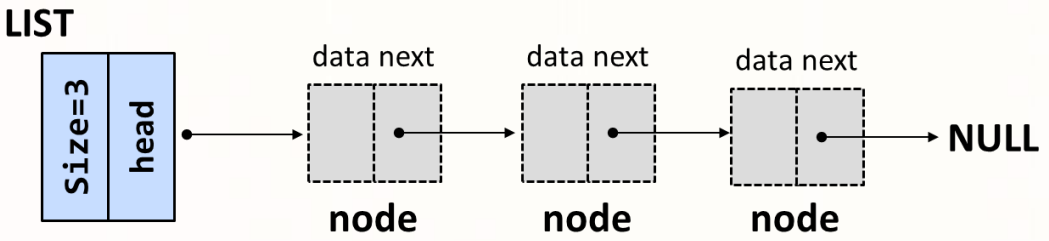
- Ευθύγραμμες Απλά Συνδεδεμένες Λίστες (εισαγωγή, εύρεση, διαγραφή)
- Ευθύγραμμες Διπλά Συνδεδεμένες Λίστες
- Σύγκριση Συνδεδεμένων Λιστών με Πίνακες

Λίστες

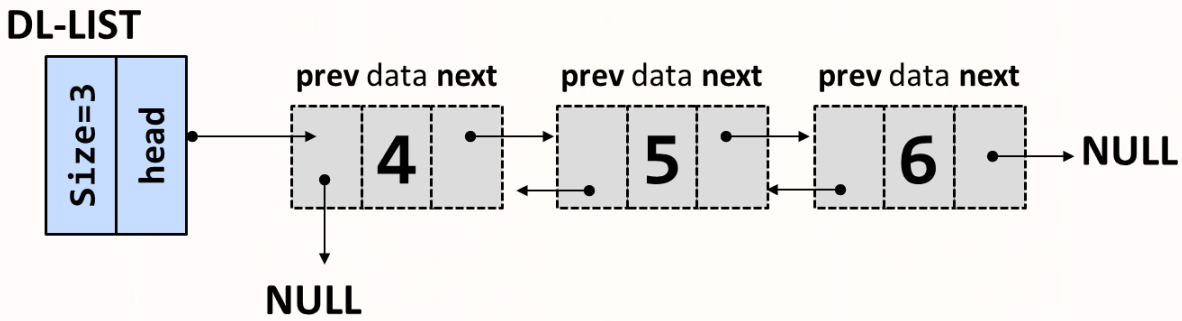
- Ο ΑΤΔ λίστα ορίζεται ως μια ακολουθία στοιχείων συνοδευόμενη από πράξεις που επιτρέπουν **εισαγωγή και εξαγωγή στοιχείων σε οποιαδήποτε θέση της λίστας. (εν αντίθεση με τις στοίβες και ουρές όπου πράξεις γίνονται μόνο στα άκρα)**
- Συνδεδεμένη λίστα
  - Συλλογή δυναμικά κατανεμημένων κόμβων
  - Κάθε κόμβος περιέχει κάποια στοιχεία
  - Κάθε κόμβος περιέχει μία ή περισσότερες συνδέσεις προς άλλους κόμβους
  - Ευμετάβλητη δομή με πολλές εφαρμογές
    - Απλή εισαγωγή και αφαίρεση στοιχείων
    - Δεν επιτρέπει τυχαία προσπέλαση κόμβων (π.χ., εύρεση του k-οστού απαιτεί προσπέλαση των πρώτων k στοιχείων)
  - Εναλλακτική λύση σε σχέση με πίνακα
    - Η καλύτερη λύση εξαρτάται από το πρόβλημα

Συνδεδεμένες Λίστες: Τύποι

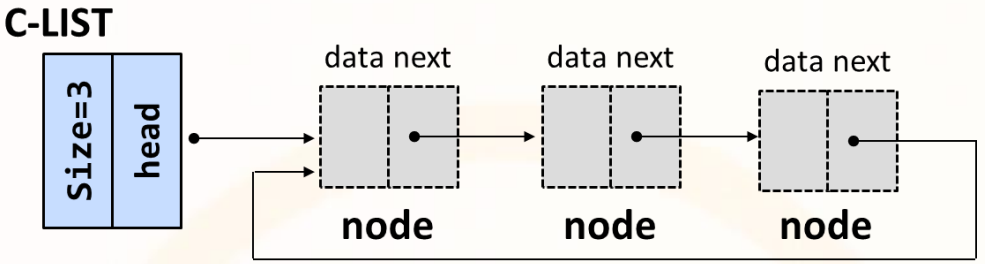
- Μονής σύνδεσης



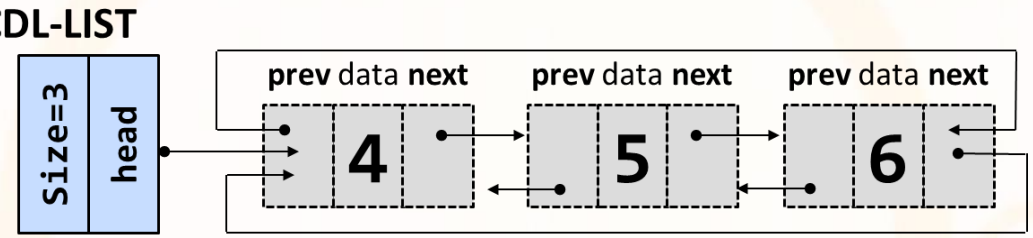
- Διπλής (ή πολλαπλής) σύνδεσης



- Κυκλικές μονής σύνδεσης



- Κυκλικές διπλής (ή πολλαπλής) σύνδεσης



Συνδεδεμένες Λίστες: Βασικές Πράξεις

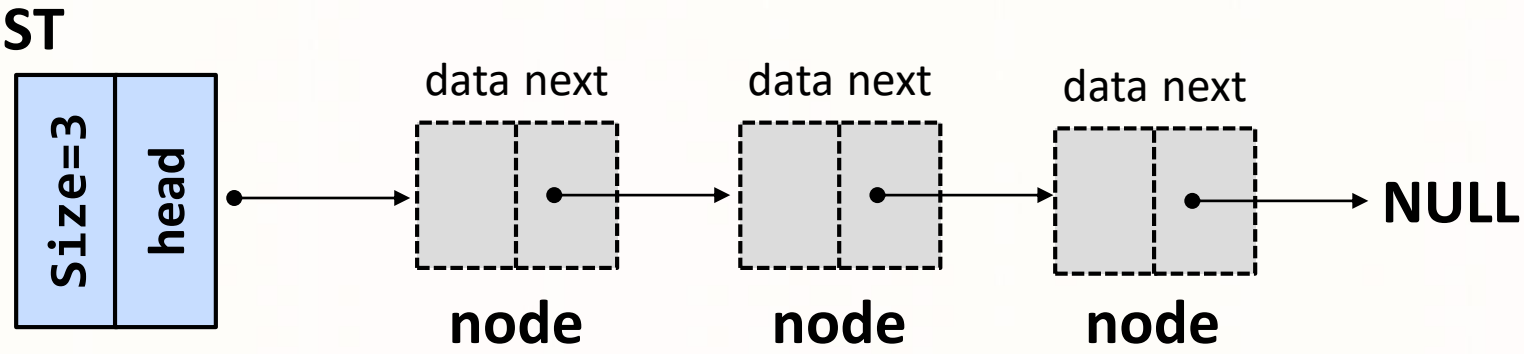
- Κάποιες τυπικές πράξεις των λιστών είναι οι πιο κάτω:

insert(L, x)	εισήγαγε το στοιχείο x στη λίστα L
delete(L, x)	διέγραψε το στοιχείο x από τη λίστα L
concatenate(L_1, L_2)	δημιούργησε μια νέα λίστα που περιέχει τα στοιχεία της λίστας L_1 ακολουθούμενα από τα στοιχεία της L_2
access(L, i)	επέστρεψε το i-οστό στοιχείο της L
sublist (L,i, j)	επέστρεψε τη λίστα που ξεκινά από το i-οστό και τελειώνει στο j-οστό στοιχείο της L.
insertAfter(L, x, i)	εισήγαγε το x μετά από το i-οστό στοιχείο της L.
delete_i(L, i)	αφαίρεσε το i-οστό στοιχείο της L

- ...και πολλές άλλες ανάλογα με τις προδιαγραφές του προγράμματος, π.χ., insertSorted

Ευθύγραμμες Απλά Συνδεδεμένες Λίστες

- Κάθε κόμβος περιέχει έναν κόμβο (ως σύνδεσμο προς τον επόμενό του)



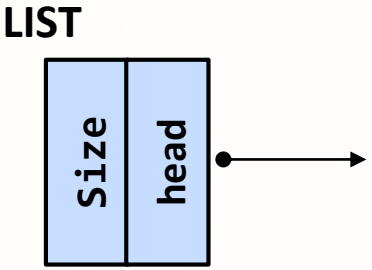
- Αυτοαναφορικές (self-referent) δομές
- Παριστάνει μία ακολουθία στοιχείων
- Πιθανή Υλοποίηση για τους κόμβους
  - E obj: για ανάθεση οποιουδήποτε τύπου αντικειμένου data
  - Ο κατασκευαστής αρχικοποιεί όλες τις τιμές.
  - Παράδειγμα δημιουργία ενός καινούριου κόμβου με ακέραιους για δεδομένα:

```
ListNode<Integer> newNode =
    new ListNode<Integer>(1, null);
```

```
private class ListNode<E> {

  private E obj;

  private ListNode<E> next;

  ListNode(E obj,
           ListNode<E> next) {
    this.obj = obj;
    this.next = next;
  }

  public E getElement(){
    return this.obj;
  }
}
```

Ευθύγραμμες Απλά Συνδεδεμένες Λίστες (συν.)

- Υλοποίηση Δομής Λίστας
  - head: δείχνει στον πρώτο κόμβο
  - size: το πλήθος των κόμβων



  - Τι συμβαίνει αν πρέπει να συγκρίνουμε (π.χ., αριθμητικά) τους κόμβους;
  - Μόνο αντικείμενα που υποστηρίζουν σύγκριση θα πρέπει να είναι αποδεκτά, δηλ.

```
public class SingleList
<E extends Comparable> {...}
```

```
public class SingleList<E>
  private ListNode<E> head;
  private int size;

  public SingleList() {
    this.head = null;
    size=0;
  }

  public void makeEmpty(){
    this.head = null;
    this.size=0;
  }

  public boolean isEmpty(){
    return this.size==0;
  }

  public int size() {
    return this.size;
  }
```

Ευθ.Απλ.Συνδ.Λισ.: Εισαγωγή στοιχείου

- Η εισαγωγή σε συνδεδεμένη λίστα μπορεί να γίνει με διάφορους τρόπους:
  - Εισαγωγή στην αρχή (παρόμοια με στοίβα)
  - Εισαγωγή στο τέλος (παρόμοια με ουρά)
  - Εισαγωγή ταξινομημένα
  - Εισαγωγή στη θέση i

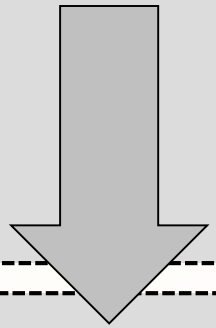
- Δεδομένου κάποιού κόμβου tmpNode, η εισαγωγή του καινούριου κόμβου newNode μετά τον tmpNode:
  - `newNode.next= tmpNode.next;`
    το `tmpNode.next` μπορεί να μην δείχνει κάπου (δηλ., `null`)
  - `tmpNode.next= newNode;`

Ευθ.Απλ.Συνδ.Λισ.: Εισαγωγή στοιχείου (συν.)

- Εισαγωγή κόμβου με στοιχείο obj πάντα στην αρχή (παρόμοια με Στοίβα):

```
public void insertFront(E obj);
```

```
public void insertFront(E obj) {
  ListNode<E> newNode = new ListNode<E>(obj, head);
  this.head = newNode;
  size+=1;
}
```



```
ListNode(E obj, ListNode<E> next) {
  this.obj = obj;
  this.next = next;
}
```

Ευθ.Απλ.Συνδ.Λισ.: Εισαγωγή στοιχείου (συν.)

- Εισαγωγή κόμβου με στοιχείο obj πάντα στο τέλος (παρόμοια με Ουρά):

```
public void insertLast(E obj);
```

```
public void insertLast(E obj) {
  ListNode<E> newNode = new ListNode<E>(obj, null);
  ListNode<E> tmp = this.head;

  while(tmp.next!=null){  // Έυρεση του
    tmp = tmp.next;       // τελευταίου
  }                       // στοιχείου

  tmp.next= newNode;      // Ανάθεση νέου στοιχείου

  size+=1;
}
```

Ευθ.Απλ.Συνδ.Λισ.: ΑΣΚΗΣΕΙΣ

- **Άσκηση 1**: Εισαγωγή κόμβου με στοιχείο obj ΤΑΞΙΝΟΜΗΜΕΝΑ:
  ```
  public void insertSorted(E obj);
  ```

- **Άσκηση 2**: Εισαγωγή κόμβου με στοιχείο obj σε συγκεκριμένη θέση
  ```
  public void insertAt(E obj, int i);
  ```

Ευθ.Απλ.Συνδ.Λισ.: Αναζήτηση

- Εντοπισμός κάποιου τυχαίου ή του n-οστού κόμβου
  - Δεν υπάρχει άμεση μέθοδος
  - Αναγκαστικά διατρέχουμε τη λίστα
  - Π.χ. αν το `tmp` δείχνει στην κεφαλή της λίστας, τότε:

```
while( tmp!=null ){
        ...
        tmp = tmp.next;
}
ή
for( int i=0; i<size(); i++)
        ...
        tmp = tmp.next; //μπορεί να προστεθεί και στο for
}
```

- Συνηθισμένα λάθη κατά την επεξεργασία λιστών
  - Χρήση αόριστης αναφοράς (προς κανένα αντικείμενο (null))
  - Χρήση αναφοράς προς λάθος αντικείμενο

Ευθ.Απλ.Συνδ.Λισ.: Αναζήτηση (συν.)

- Πιο κάτω ορίζονται κάποιες χρήσιμες πράξεις.
- Εύρεση Κόμβου με συγκεκριμένο στοιχείο:
  public boolean existsNode(E obj);

```
public boolean existsNode(E obj){
  ListNode<E> tmp = this.head;

  while( tmp!=null ){
    if( tmp.obj.equals(obj))
      return true;
    tmp = tmp.next;
  }
  return false;
}
```

Το tmp είναι ένα αντίγραφο της διεύθυνσης στην οποία δείχνει στο πρώτο στοιχείο της λίστας (δηλ., head)

Ευθ.Απλ.Συνδ.Λισ.: Αναζήτηση (συν.)

- Με παρόμοιο τρόπο μπορούμε να ορίσουμε διαδικασία:
  `public ListNode<E> findNode(E obj);`
  που επιστρέφει δείκτη προς κόμβο της λίστας που περιέχει το στοιχείο obj, αν υπάρχει.

```
public ListNode<E> findNode(E obj){
  ListNode<E> tmp = this.head;

  while( tmp!=null ){
    if( tmp.obj.equals(obj))
        return tmp;
    tmp = tmp.next;
  }
  return null;
}
```

Ευθ.Απλ.Συνδ.Λισ.: Αναζήτηση (συν.)

- Με χρήση for η μέθοδος
  `public ListNode<E> findNode(E obj);`

```
public ListNode<E> findNode(E obj){
  ListNode<E> tmp = this.head;

  for( int i=0; i<size(); i++){

    if( tmp.obj.equals(obj))
        return tmp;
    tmp = tmp.next;
  }
  return null;
}
```

Ευθ.Απλ.Συνδ.Λισ.: Διαγραφή στοιχείου

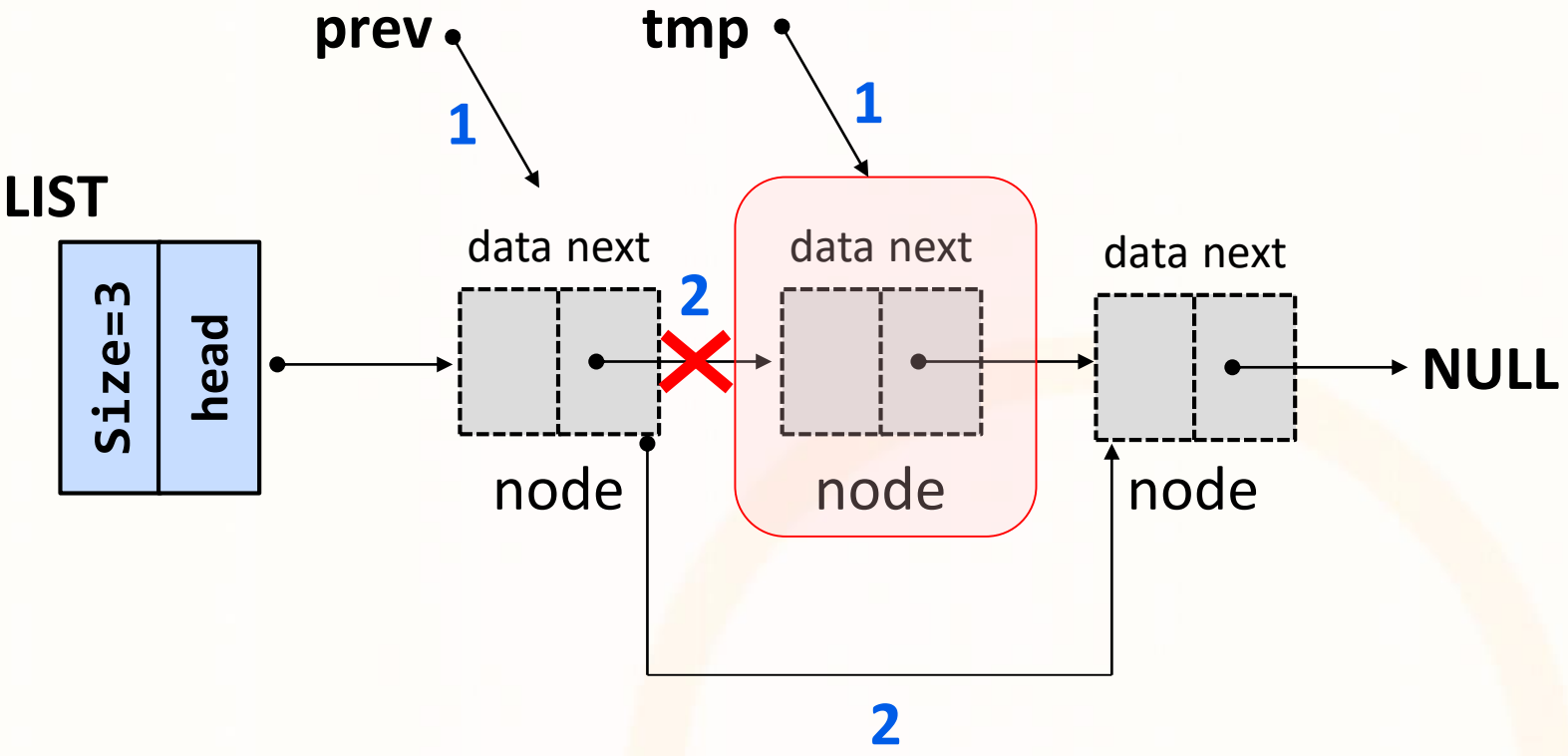
- Συμμετρικά με την εισαγωγή σε συνδεδεμένη λίστα, η διαγραφή μπορεί να γίνει με διάφορους τρόπους:
  - Διαγραφή του πρώτου κόμβου (παρόμοια με στοίβα)
  - Διαγραφή του τελευταίου κόμβου
  - Διαγραφή συγκεκριμένου στοιχείου
  - Διαγραφή ι-οστού στοιχείου

- Δεδομένου κάποιού κόμβου prev, η διαγραφή κάποιου κόμβου tmp μετά τον prev:
  - `prev.next= tmp.next;`
    το `tmp.next` μπορεί να μην δείχνει κάπου (δηλ., `null`)

- Συνηθισμένα λάθη κατά την διαγραφή κόμβων
  - Διαγραφή του πρώτου στοιχείου (πρέπει να αλλαχθεί το head)
  - Προσπάθεια πρόσβασης του next από null κόμβο

Ευθύγραμμες Απλά Συνδεδεμένες Λίστες (συν.)

- Εξαγωγή Κόμβου με συγκεκριμένη πληροφορία “obj”:
  `public void delete(E obj);`
- Χρήση δύο δεικτών (prev, tmp) για ολοκλήρωση του αλγόριθμου



Ευθύγραμμες Απλά Συνδεδεμένες Λίστες (συν.)

- Εξαγωγή Κόμβου με συγκεκριμένη πληροφορία “obj”:
  public void delete(E obj);

```
public void delete(E obj) {
  if (!isEmpty()) {
    ListNode<E> tmp = this.head;
    ListNode<E> prev = tmp;

    //if first node is to be
deleted

if(tmp.getElement().equals(obj)){
      this.head = this.head.next;
      size-=1;
    }
    ...
```

```
    ...
    else {
      while(tmp!=null) {

if(tmp.getElement().equals(obj)){
          prev.next = tmp.next;
          size-=1;
          break;
        }
        prev = tmp;
        tmp = tmp.next;
      }
    }
  }
}
```

Ευθ.Απλ.Συνδ.Λισ.: ΑΣΚΗΣΕΙΣ

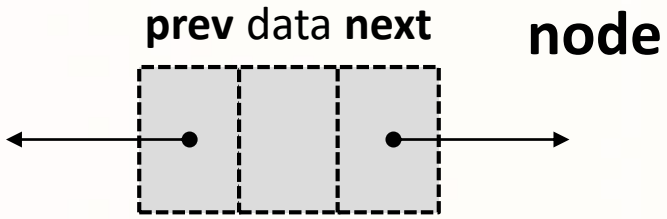
- **Άσκηση 3**: Διαγραφή κόμβου στην αρχή:
```
public void deleteFirst();
```

- **Άσκηση 4**: Διαγραφή κόμβου στο τέλος:
```
public void deleteLast();
```

- **Άσκηση 5**: Διαγραφή κόμβου στη θέση i:
```
public void deleteAt(int i);
```

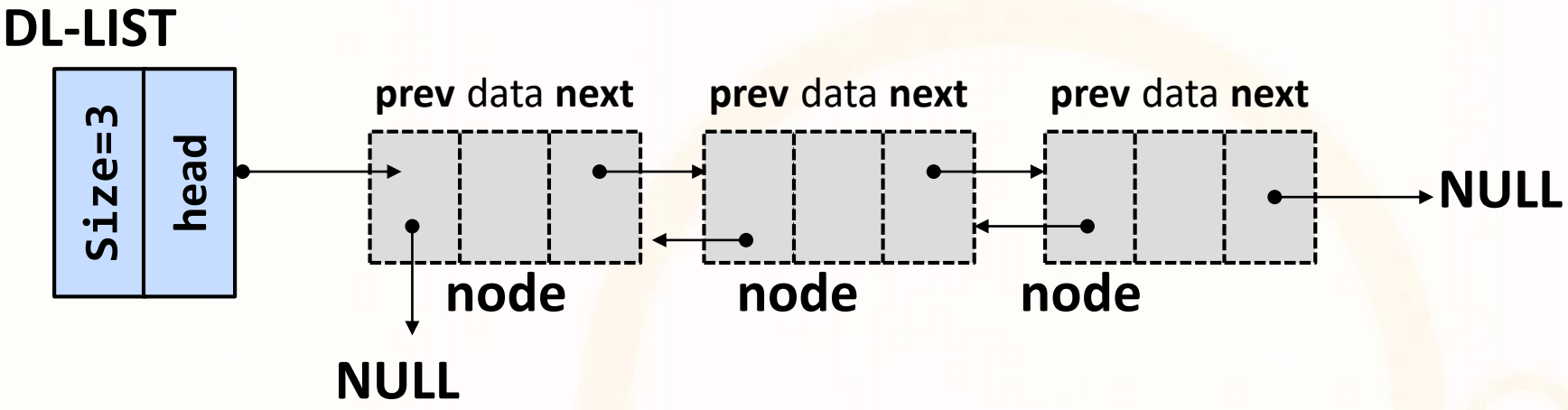
Ευθύγραμμες Διπλά Συνδεδεμένες Λίστες

- **Διπλά συνδεδεμένη λίστα (doubly-linked list)** ονομάζεται μια λίστα κάθε κόμβος της οποίας κρατά πληροφορίες και για τον επόμενο και για τον προηγούμενο κόμβο:



- Με αυτό τον τρόπο δίνεται η ευχέρεια μετακίνησης μέσα στη λίστα και προς τις δύο κατευθύνσεις.

- Παράδειγμα Λίστας:



Ευθύγραμμες Διπλά Συνδεδεμένες Λίστες (συν.)

- Πλεονεκτήματα/Μειονεκτήματα
  - 2 δείκτες σε κάθε κόμβο
  - Κάθε κόμβος δείχνει και στον προηγούμενο
  - Κίνηση προς δύο κατευθύνσεις
  - **Ευκολότερη εισαγωγή και διαγραφή**
    - Εισαγωγή: αρκεί να έχουμε δείκτη προς προηγούμενο ή επόμενο κόμβο
    - Διαγραφή: αρκεί ο προς διαγραφή κόμβος
  - **Μεγαλύτερο κόστος συντήρησης**
    - Διπλάσιοι σύνδεσμοι προς ενημέρωση
  - **Μεγαλύτερο κόστος μνήμης**
    - Διπλάσιοι σύνδεσμοι προς αποθήκευση

Ευθ. Διπλά Συνδ. Λίστ.: Υλοποίηση

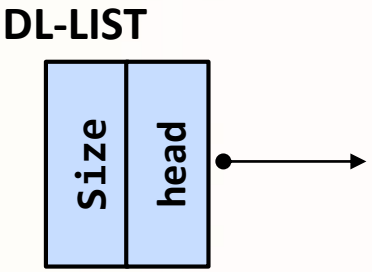
- Πιθανή Υλοποίηση για τους κόμβους
  - Παρόμοια με απλά συνδεδεμένη λίστα
  - **E obj**: για ανάθεση οποιουδήποτε τύπου αντικειμένου data
  - **prev**: αναφορά στον προηγούμενο κόμβο
  - **next**: αναφορά στον επόμενο κόμβο
  - Παράδειγμα δημιουργία ενός καινούριου κόμβου με ακέραιους για δεδομένα:

```
ListNode<Integer> newNode =
     new ListNode<Integer>(1, null, null);
```

```
private class ListNode<E> {

  private E obj;

  private ListNode<E> prev;
  private ListNode<E> next;

  ListNode(E obj) {
    this.obj = obj;
    this.prev = null;
    this.next = null;
  }

  public E getElement(){
    return this.obj;
  }
}
```

Ευθ. Διπλά Συνδ. Λίστ.: Υλοποίηση (συν.)

- Υλοποίηση Δομής Λίστας
  - head: δείχνει στον πρώτο κόμβο
  - size: το πλήθως των κόμβων



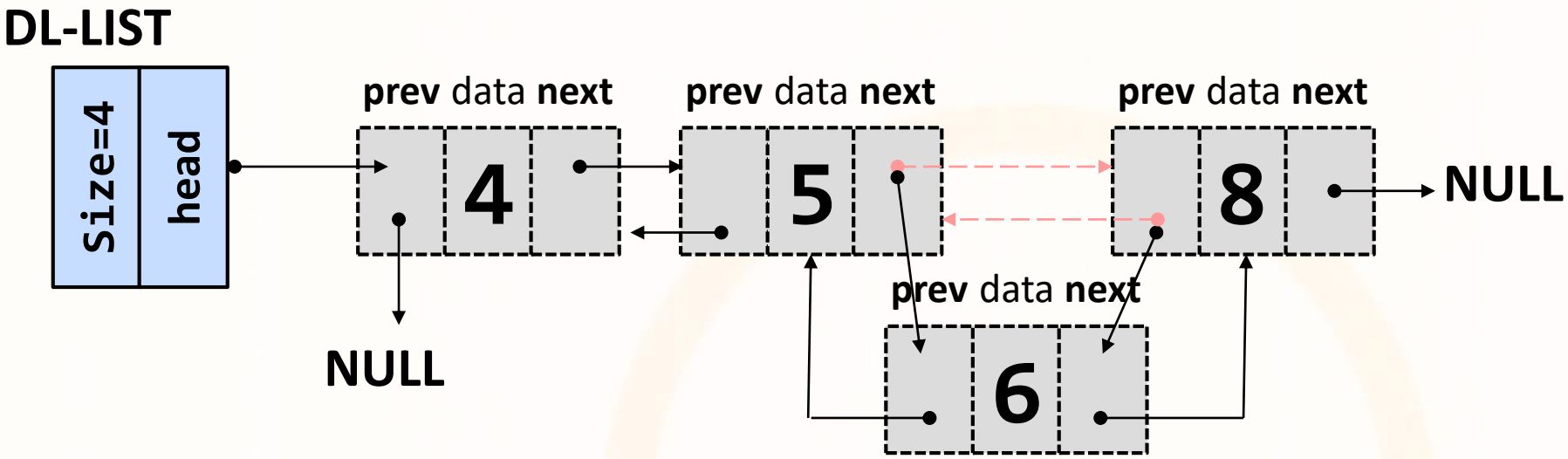
  - Τι συμβαίνει αν πρέπει να συγκρίνουμε (π.χ., αριθμητικά) τους κόμβους;
  - Μόνο αντικείμενα που υποστηρίζουν σύγκριση θα πρέπει να είναι αποδεκτά, δηλ.

```
public class DoubleList
<E extends Comparable> {...}
```

```
public class DoubleList<E>
  private ListNode<E> head;
  private int size;

  public SingleList() {
    this.head = null;
    size=0;
  }

  public void makeEmpty(){
    this.head = null;
    this.size=0;
  }

  public boolean isEmpty(){
    return this.size==0;
  }

  public int size() {
    return this.size;
  }
```

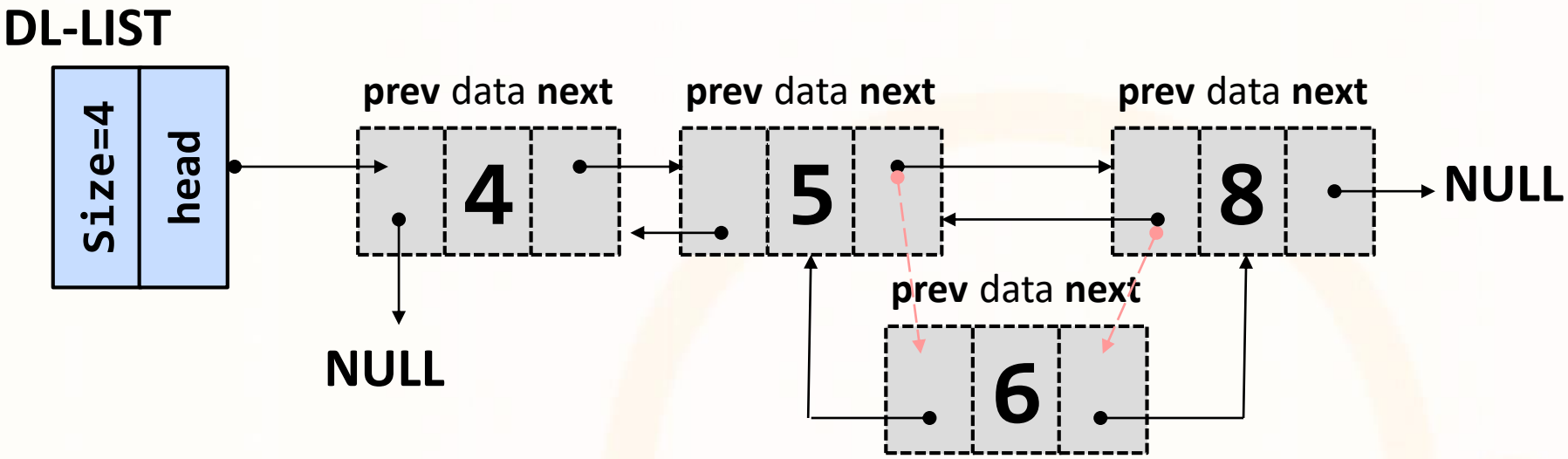
Ευθ. Διπλά Συνδ. Λίστ.: Εισαγωγή Στοιχείου

- Προφανώς η εισαγωγή στοιχείου σε κάποιο σημείο μιας διπλά συνδεδεμένης λίστας περιέχει κάποια **επιπλέον πολυπλοκότητα** από την εισαγωγή σε μια απλά συνδεδεμένη λίστα.
- Αυτό γιατί κάθε νέος κόμβος πρέπει να συνδεθεί και **με τον επόμενο** και με τον **προηγούμενο κόμβο στη λίστα**.
- Παράδειγμα εισαγωγής του στοιχείου 6 μετά το 5 στην πιο κάτω λίστα:



Ευθ. Διπλά Συνδ. Λίστ.: Διαγραφή Στοιχείου

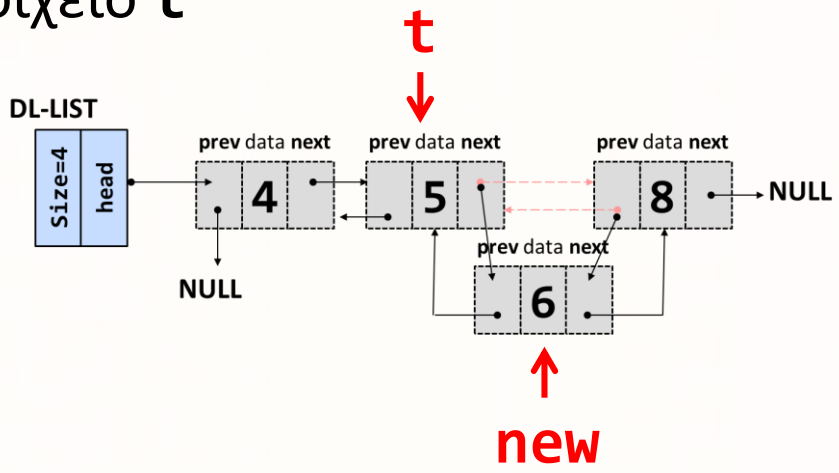
- Αντίθετα, η διαγραφή στοιχείου σε κάποιο σημείο μιας διπλά συνδεδεμένης λίστας είναι **πιο εύκολη** από την εισαγωγή σε μια απλά συνδεδεμένη λίστα.
- Αυτό γιατί δεν χρειάζονται δείκτες σε άλλους κόμβους
- Παράδειγμα εισαγωγής του στοιχείου 6 μετά το 5 στην πιο κάτω λίστα:



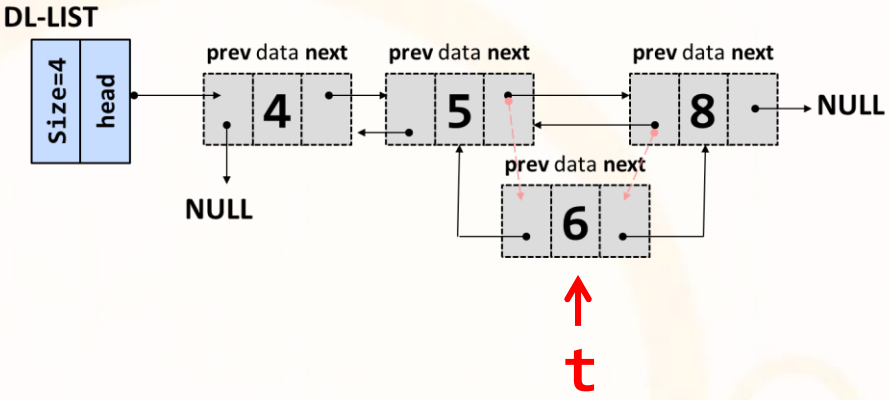
Ευθ. Διπλά Συνδ. Λίστ.: Υλοποίηση

- **Μετά την εύρεση της θέσης εισαγωγής/διαγραφής**

- **Εισαγωγή** στοιχείου **new** μετά το στοιχείο **t**
  - `new.next = t.next;`
  - `new.prev = t;`
  - `t.next.prev = new;`
  - `t.next = new;`



- **Διαγραφή** στοιχείου **t**
  - `t.next.prev = t.prev;`
  - `t.prev.next = t.next;`



Πίνακες vs. Συνδεδεμένες Λίστες

- Όταν υλοποιούμε λίστες με πίνακες χρειάζεται να γνωρίζουμε το μέγιστο μέγεθος της λίστας εκ των προτέρων.

- **Χρήση Χώρου**
  - **Πίνακας:** καταλαμβάνεται ο ίδιος χώρος άσχετα με τον αριθμό των στοιχείων που είναι αποθηκευμένα.
  - **Συνδεδεμένη Λίστα:** μέγεθος της λίστας × (χώρος ενός στοιχείου + χώρος ενός δείκτη)

- **Χρόνος εισαγωγής / εξαγωγής** στην αρχή
  - **Πίνακας:** Ίσως χρειαστεί να μετακινήσουμε όλα τα στοιχεία $\in \Theta(n)$
  - **Συνδεδεμένη Λίστα:** Δεν μετακινείται τίποτε $\in \Theta(1)$

- **Χρόνος εύρεσης k-οστού** στοιχείου
  - **Πίνακας:** Κατευθείαν $\in \Theta(1)$
  - **Συνδεδεμένη Λίστα:** Χρειάζεται να περάσουμε από Κ άλλους κόμβους $\in \Theta(k)$