Διάλεξη 12: Λίστες – Υλοποίηση & Εφαρμογές

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

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

Διδάσκων: Παναγιώτης Ανδρέου

Λίστες

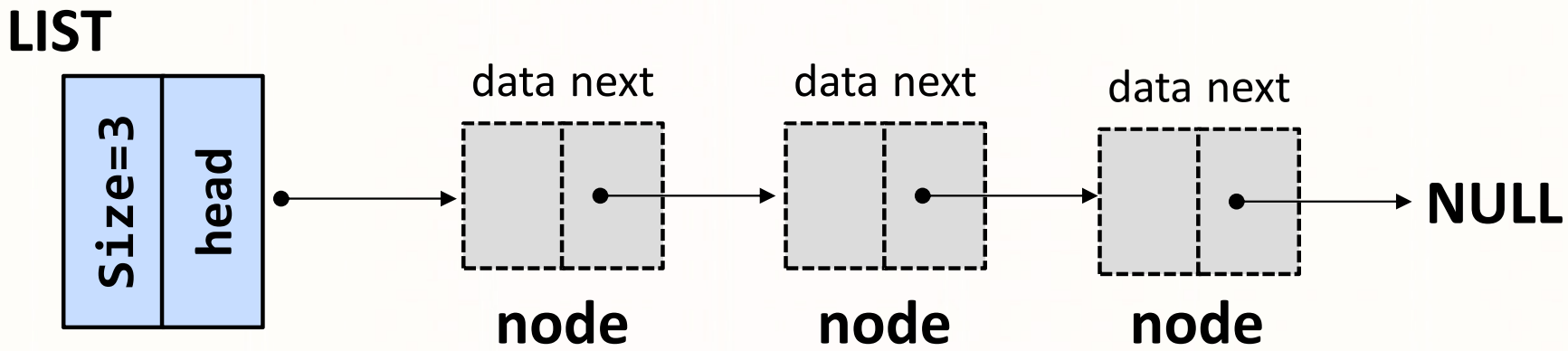
- Ο ΑΤΔ λίστα ορίζεται ως μια ακολουθία στοιχείων συνοδευόμενη από πράξεις που επιτρέπουν **εισαγωγή και εξαγωγή στοιχείων σε οποιαδήποτε θέση της λίστας. (εν αντίθεση με τις στοίβες και ουρές όπου πράξεις γίνονται μόνο στα άκρα)**
- Κάποιες βασικές πράξεις των λιστών είναι οι πιο κάτω:

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.
Insert_After(L, x, i)	εισήγαγε το x μετά από το i-οστό στοιχείο της L.
Delete(L, i)	αφαίρεσε το i-οστό στοιχείο της L

- Στη συνέχεια θα μελετήσουμε υλοποιήσεις του ΑΤΔ με δυναμική χορήγηση μνήμης.

Ευθύγραμμες Απλά Συνδεδεμένες Λίστες

- Μία λίστα μπορεί να υλοποιηθεί ως μια συνδεδεμένη λίστα (με παρόμοιο τρόπο όπως μια στοίβα).



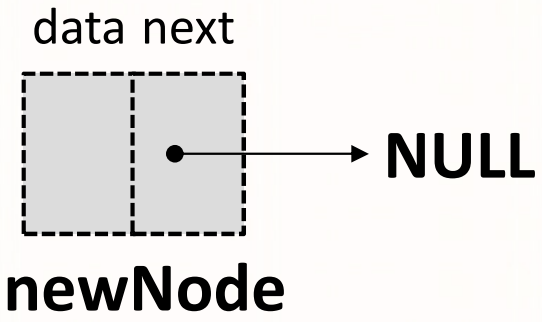
- Πιθανές δηλώσεις κόμβων είναι (χρησιμοποιείται `int` αντί για `type`):

```
typedef struct node {
   int          data;
   struct node *next;
} NODE;
```

```
typedef struct list {
   NODE  *head;
   int    size;
} LIST;
```

Ευθύγραμμες Απλά Συνδεδεμένες Λίστες (συν.)

- Βοηθητική συνάρτηση: Δημιουργία κόμβου
  `NODE* createListNode(int x)`




```
NODE* createListNode(int x){
  NODE *newNode = NULL;
  newNode = (NODE*) malloc( sizeof(NODE) );
  newNode->data = x;
  newNode->next = NULL;
  return newNode;
}
```

Ευθύγραμμες Απλά Συνδεδεμένες Λίστες (συν.)

- Πιο κάτω ορίζονται κάποιες χρήσιμες πράξεις.
- Εύρεση Κόμβου με συγκεκριμένο στοιχείο:
  `bool existsNode(LIST *L, int x)`

```
bool existsNode(LIST *L, int x){
  NODE* tmp = L->head;

  while( tmp!=NULL ) {
    if( tmp->data == x )
      return true;
    tmp = tmp->next;
  }
  return false;
}
```

Το p είναι ένα αντίγραφο της διεύθυνσης στην οποία δείχνει το L->head

Ευθύγραμμες Απλά Συνδεδεμένες Λίστες (συν.)

- Με παρόμοιο τρόπο μπορούμε να ορίσουμε διαδικασία:
  `NODE* findNode(LIST *L, int x);`
  που επιστρέφει δείκτη προς κόμβο της λίστας που περιέχει το στοιχείο x, αν υπάρχει.

```
NODE* findNode(LIST *L, int x){
  NODE* tmp = L->head;

  while( tmp!=NULL ){
    if( tmp->data == x )
      return tmp;
    tmp = tmp->next;
  }
  return NULL;
}
```

Ευθύγραμμες Απλά Συνδεδεμένες Λίστες (συν.)

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

```
void insertListFront(LIST *L, int x);
```

```
void insertListFront(LIST *L, int x){
  NODE *newNode = createListNode(x);
  newNode->next = L->head;
  L->head = newNode;
  L->size++;
}
```

Ευθύγραμμες Απλά Συνδεδεμένες Λίστες (συν.)

- Εισαγωγή κόμβου με στοιχείο x ΤΑΞΙΝΟΜΗΜΕΝΑ:

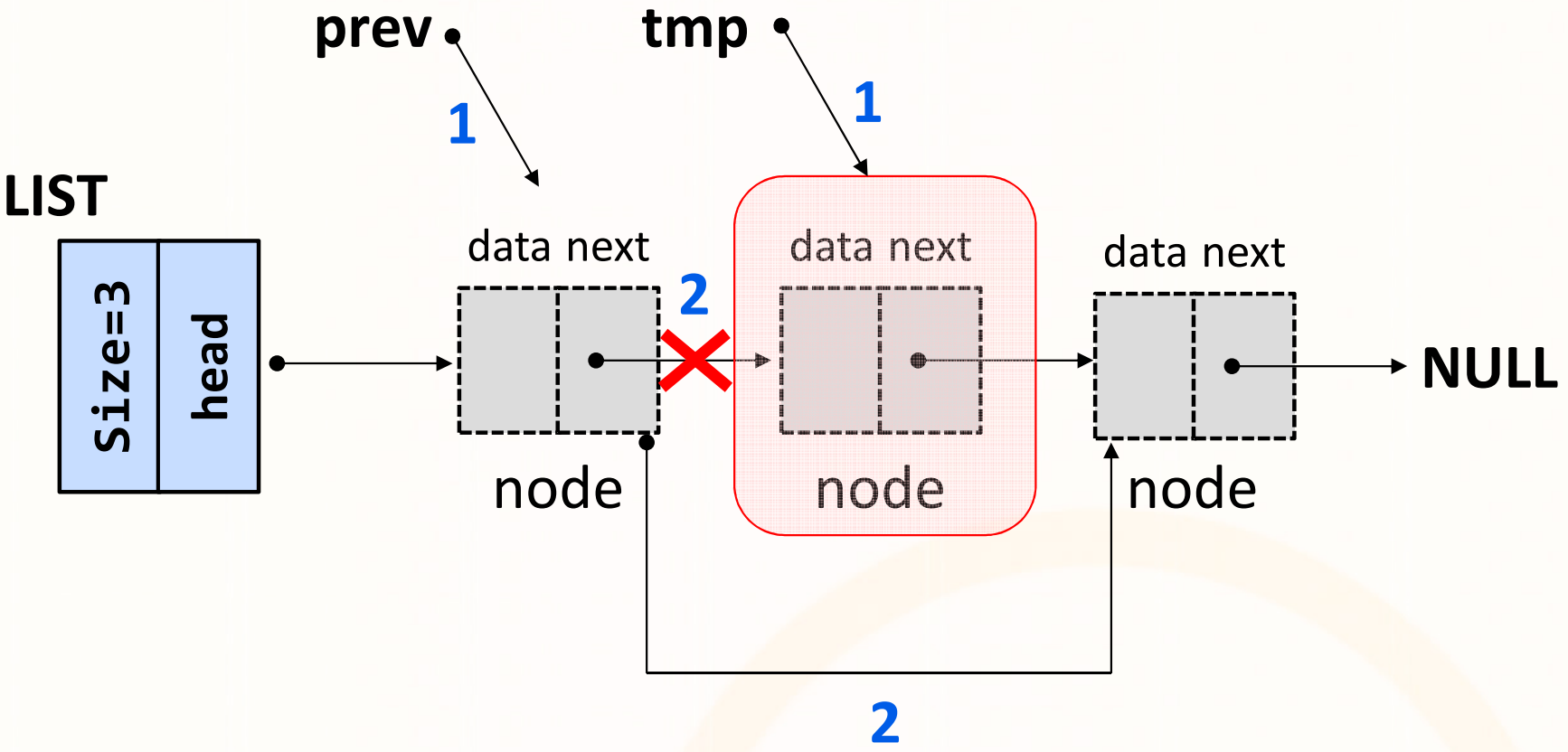
```
void insertListSorted (LIST *L, int x);
```

```
void insertListSorted (
                       LIST *L,
                       int x){
  NODE *newNode =
            createListNode(x);
  NODE *tmp = L->head;
  if( L->size==0 ) {
     L->head = newNode;
  }
  else {
    if(tmp->data>newNode-
>data){
      newNode->next=tmp; L-
>head = newNode;
    }
```

```
  else {
    while( (tmp != NULL) ) {
      if(tmp->next == NULL)
        break;
      if(tmp->next-
>data>newNode->data)
        break;
      tmp = tmp->next;
    }
    newNode->next = tmp->next;
    tmp->next = newNode;
    }
  }
  L->size++;
}
```

Ευθύγραμμες Απλά Συνδεδεμένες Λίστες (συν.)

- Εξαγωγή Κόμβου με συγκεκριμένη πληροφορία “x”:
  `void deleteNode(LIST *L, int x)`



Ευθύγραμμες Απλά Συνδεδεμένες Λίστες (συν.)

- Εξαγωγή Κόμβου με συγκεκριμένη πληροφορία “x”:
  `void deleteNode(LIST *L, int x)`

```
void deleteNode(LIST *L,
                int x){
  NODE* tmp = L->head;
  NODE* prev = tmp;

  if(!IsEmpty(L)){
    if (L->head->data == x){
      L->head = L->head->next;
      L->size--;
      free(tmp);
    }
    …
```

```
    else {
      while( tmp!=NULL ){
        if( tmp->data == x ){
          prev->next=tmp->next;
          L->size--;
          free(tmp);
          break;
        }
        prev = tmp;
        tmp = tmp->next;
      }
    }
  }
}
```

Πίνακες vs. Συνδεδεμένες Λίστες

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

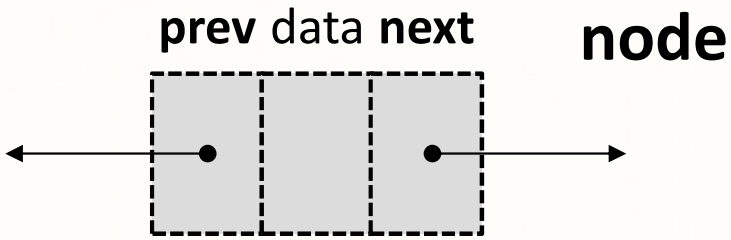
- **Χρήση Χώρου**
  - **Πίνακας:** καταλαμβάνεται ο ίδιος χώρος άσχετα με τον αριθμό των στοιχείων που είναι αποθηκευμένα.
  - **Συνδεδεμένη Λίστα:** μέγεθος της λίστας × (χώρος ενός στοιχείου + χώρος ενός δείκτη)

- **Χρόνος εισαγωγής / εξαγωγής** στην αρχή
  - **Πίνακας:** Ίσως χρειαστεί να μετακινήσουμε όλα τα στοιχεία
  - **Συνδεδεμένη Λίστα:** Δεν μετακινείται τίποτε

- **Χρόνος εύρεσης k-οστού** στοιχείου
  - **Πίνακας:** Κατευθείαν
  - **Συνδεδεμένη Λίστα:** Χρειάζεται να περάσουμε από Κ άλλους κόμβους

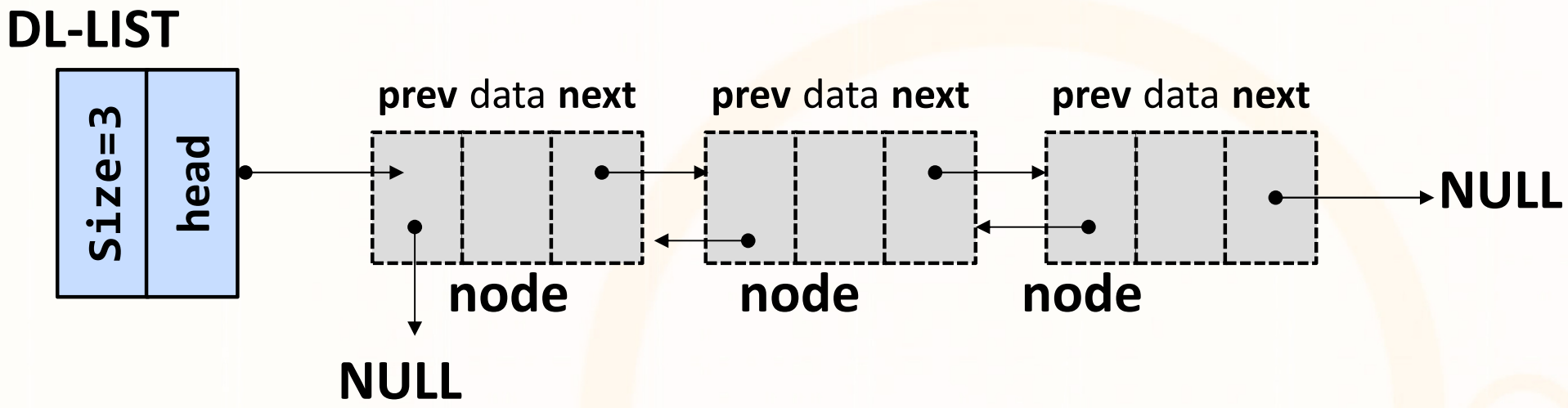
Ευθύγραμμες Διπλά Συνδεδεμένες Λίστες

- **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:



Διπλά Συνδεδεμένες Λίστες

- Ποιες δομές χρειάζονται για υλοποίηση μιας διπλά συνδεδεμένης λίστας;

- Ένας κόμβος ορίζεται από το πιο κάτω structure:

```
typedef struct dlnode {
   int          data;
   struct node *next;
   struct node *prev;
} DLNODE;
```

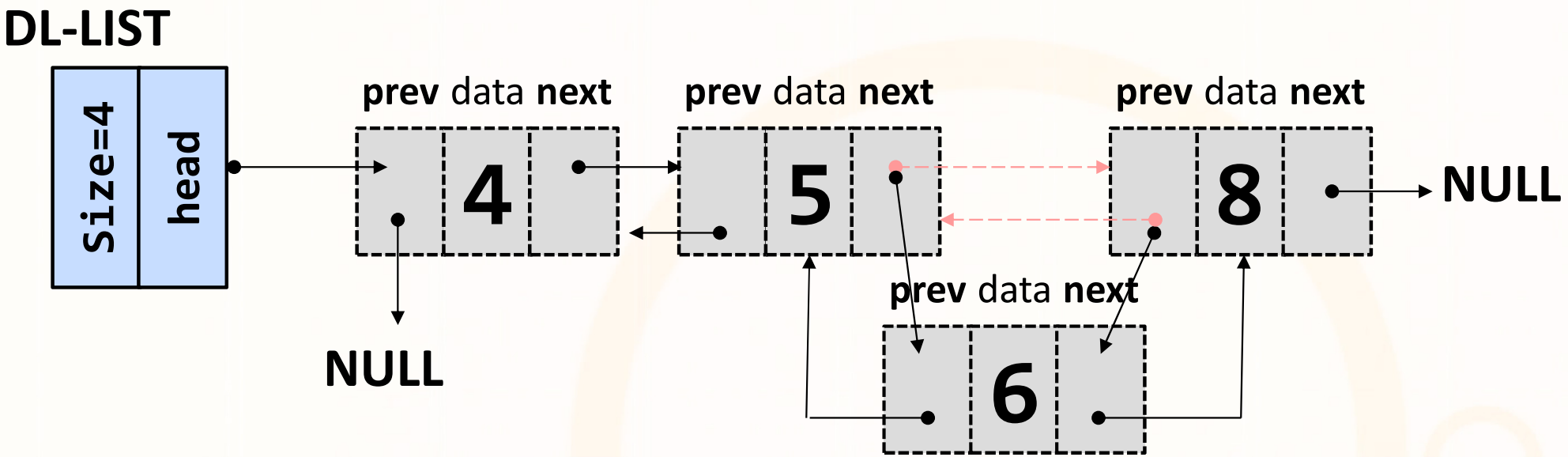
- Ο κόμβος που ορίζει τη διπλά συνδεδεμένη λίστα είναι ο ίδιος με αυτό που ορίζει μια στοίβα:

```
typedef struct list {
   DLNODE  *head;
   int      size;
} DLLIST;
```

Διπλά Συνδεδεμένες Λίστες

- 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**. Παρόμοια, κατά τις εξαγωγές στοιχείων.

- Παράδειγμα εισαγωγής του στοιχείου 6 μετά το 5στην πιο κάτω λίστα:



Η συνάρτηση put

- Να ορίσετε συνάρτηση put(l, x, y) η οποία τοποθετεί το **στοιχείο x** μετά από το **στοιχείο y** μέσα στη **λίστα l**.

```
put(DLLIST *l, int x, int y){
  if (l->head == NULL)
    printf("The list is empty, no insertion was made\n");
  else {
    putnode(l->head, x, y);
  }
}
```

Η συνάρτηση puttnode

```
// Τοποθέτηση x μετά το y ξεκινώντας από τον δείκτη p
void  putnode(DLNODE *p, int x, int y){
  DLNODE  *q;
  if (p == NULL) // reached end of list (and did not find y)
       printf("Element %d does not exist,
                  no insertion was made\n", y);
  else if ( p->data == y ) { // position found – do insertion
       q = (DLNODE *)malloc(sizeof(DLNODE));
       q->data = x;
       q->next = p->next;
       q->prev = p;
       (p->next)->prev = q;
       p->next= q;
  }
  else // position not found – search next
       putnode(p->next, x, y);
  }
}
```