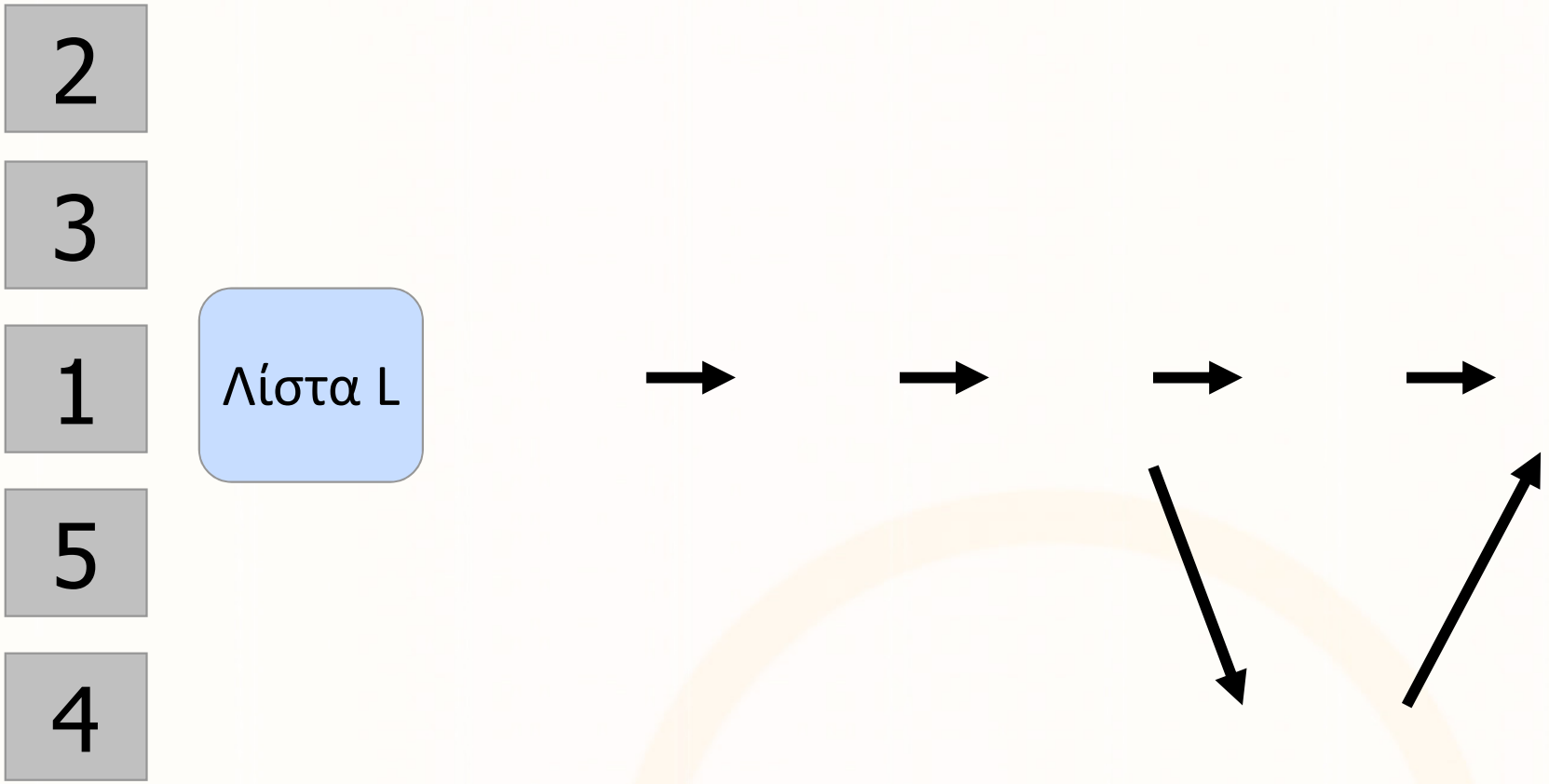
# Εργαστήριο 4: Υλοποίηση Αφηρημένου Τύπου Δεδομένων: Ταξινομημένη Λίστα

**Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:**

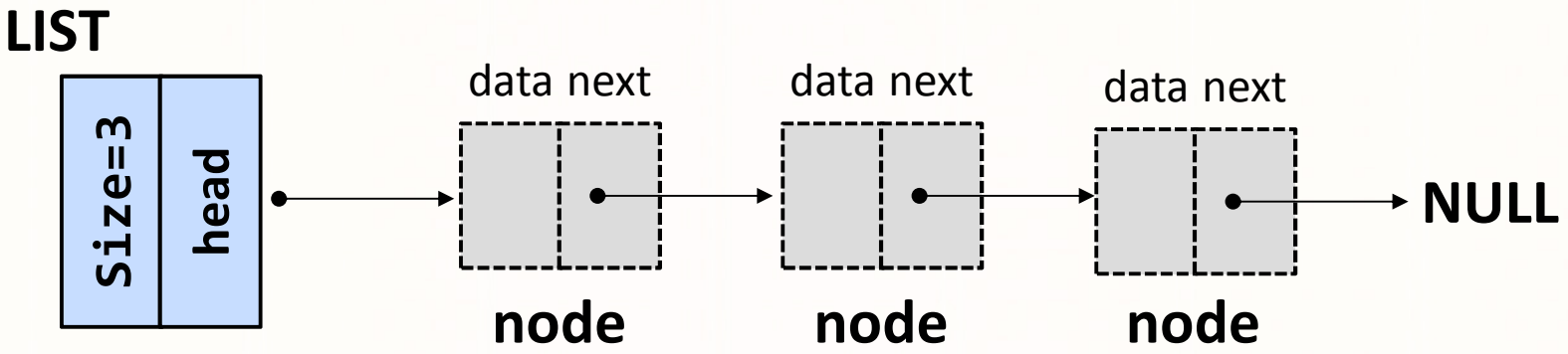
- Λίστες
- Υλοποίηση ταξινομημένης λίστας με δυναμική δέσμευση μνήμης

# Παράδειγμα ταξινομημένης λίστας



# Ευθύγραμμες Απλά Συνδεδεμένες Λίστες

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



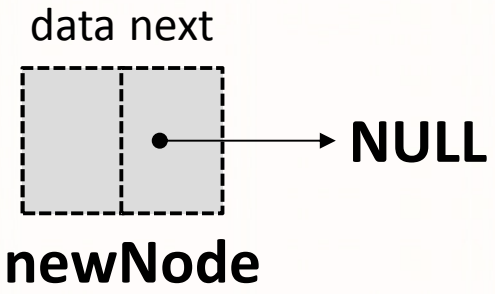

- Πιθανές δηλώσεις κόμβων είναι (χρησιμοποιείται `int` αντί για `type`):

```
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

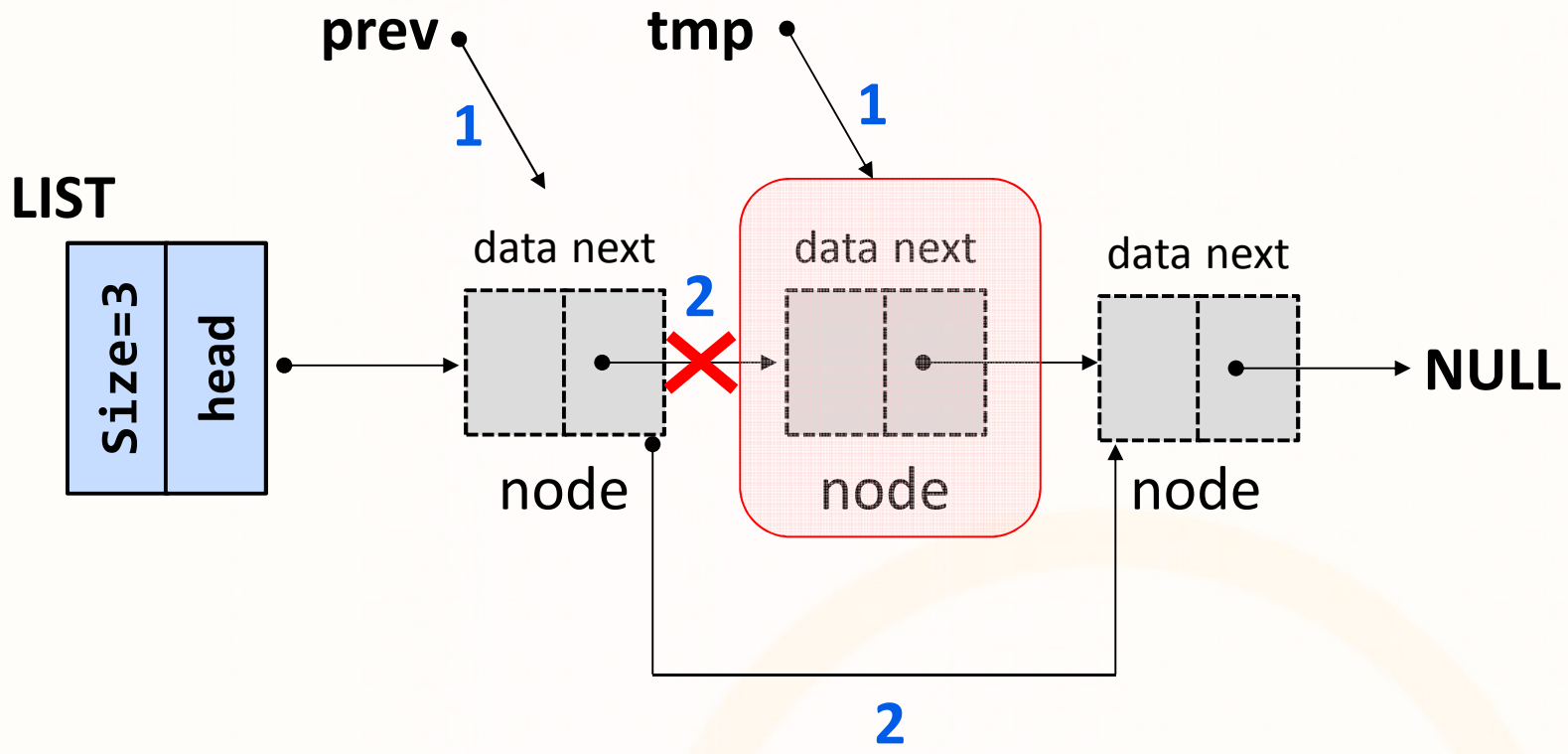
# Ευθύγραμμες Απλά Συνδεδεμένες Λίστες (συν.)

- 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.

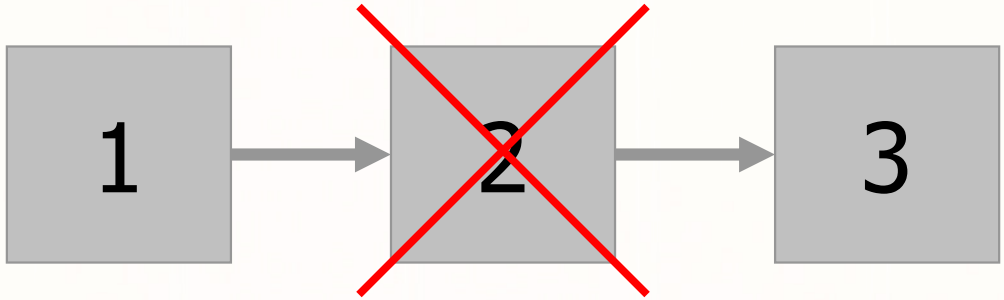
```
NODE* findNode(LIST *L, int x){
  NODE* tmp = L->head;

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

# Ευθύγραμμες Απλά Συνδεδεμένες Λίστες (συν.)

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



# Υλοποίηση της συνάρτησης Delete




**1. Ο απλός τρόπος**

- Node1→next = Node2→next
- Πιο είναι το πρόβλημα? ---
  Ο κόμβος Node2 παραμένει στην μνήμη

X

**2. Ο ΣΩΣΤΟΣ τρόπος (απελευθέρωση αχρείαστης μνήμης)**

- TempNode = Node2
- Node1→next = Node2→next
- free(TempNode)

√

# Υλοποίηση Ταξινομημένης Λίστας

## Δομές που θα χρειαστείτε:

```
//easily change the implementation from int to
other type, e.g., typedef double type;
typedef int type;

//The NODE data structure
typedef struct node {
  type          data;
  struct node *next;
} NODE;

//The LIST data structure
typedef struct list {
  NODE  *head;
  int   size;
}LIST;
```

## Συναρτήσεις που πρέπει να υλοποιήσετε

- `NODE* createListNode(type x);`
- `void makeEmptyList(LIST *L);`
- `bool isEmptyList(LIST *L);`
- `void printList(LIST *L);`
- `void insertNode(LIST *L, type x);`
- `void deleteNode(LIST *L, type x);`
- `type deleteMin(LIST *L);`
- `type deleteMax(LIST *L);`

# Λύσεις

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

# Λύσεις (συν.)

```
void makeEmptyList(LIST *L) {
   L->size = 0;
   L->head = NULL;
}
```

# Λύσεις (συν.)

```
bool IsEmpty(LIST *L) {
   return (L->size == 0);
}
```

# Λύσεις (συν.)

```
void printList(LIST *L) {
  NODE* tmp = L->head;
  for(int i=0; i<L->size; i++){
    printf("%d ", tmp->data);
    tmp = tmp->next;
  }
  printf("\n");
}
```

# Λύσεις (συν.)

```
void insertNode(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++;
}
```

# Λύσεις (συν.)

```
void deleteNode(LIST *L, int
x){
  NODE* tmp = L->head;
  NODE* previous = 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 ){
          previous->next = tmp-
>next;
          L->size--;
          free(tmp);
          break;
        }
        previous = tmp;
        tmp = tmp->next;
      }
    }
  }
}
```

# Λύσεις (συν.)

```
type deleteMin(LIST *L){
  type result=-1;
  NODE* tmp = L->head;

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

# Λύσεις (συν.)

```
type deleteMax(LIST *L){
  type result =-1;
  NODE* tmp   = L->head;
  NODE* prev  = L->head;
  if(!IsEmpty(L)){
    if(L->size==1) {
      result = tmp->data;
      free(tmp);
      L->head = NULL;
    }
    else{
      while( (tmp->next)!=NULL){
        prev = tmp;
        tmp = tmp->next;
      }
      result = tmp->data;
      free(tmp);
      prev->next=NULL;
    }
    L->size--;
    return result;
  }
  return -1;
}
```