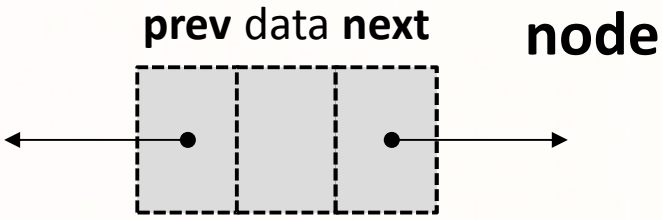
Εργαστήριο 5: Υλοποίηση Αφηρημένου Τύπου Δεδομένων: Διπλά Συνδεδεμένη Λίστα

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

- Διπλά Συνδεδεμένες Λίστες
- Υλοποίηση Διπλά Συνδεδεμένης Λίστας με δυναμική δέσμευση μνήμης

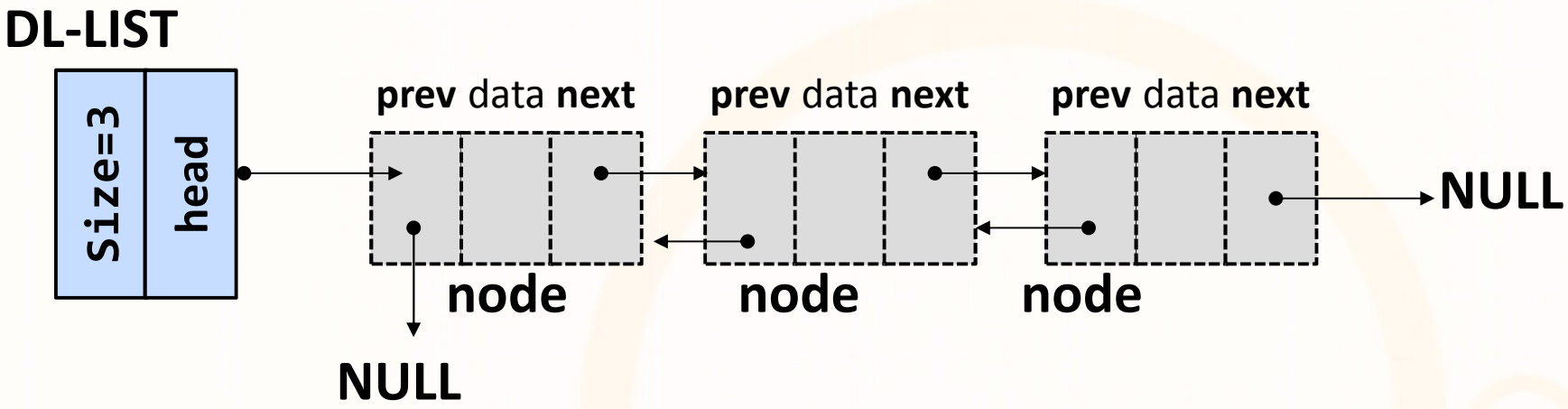
Διπλά Συνδεδεμένες Λίστες

- **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 node {
   int          data;
   struct node *next;
   struct node *prev;
} NODE;
```

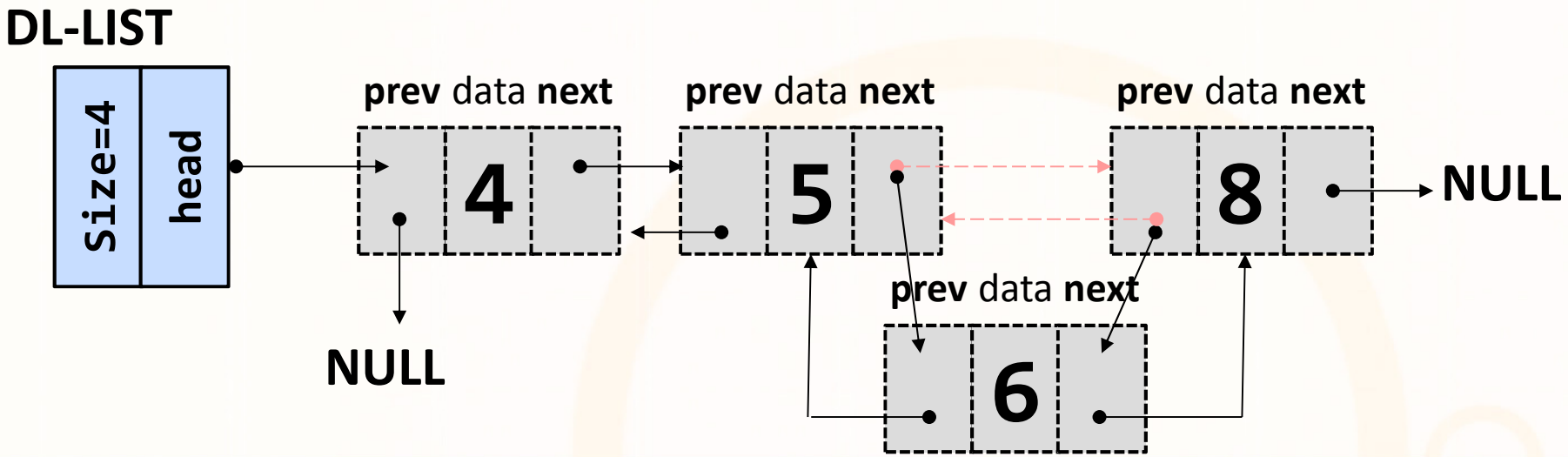
- Ο κόμβος που ορίζει τη διπλά συνδεδεμένη λίστα είναι ο ίδιος με αυτό που ορίζει μια στοίβα:

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

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.

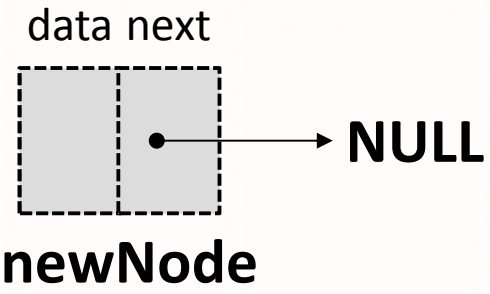
- Παράδειγμα εισαγωγής του στοιχείου 6 μετά το 5στην πιο κάτω λίστα:



Διπλά Απλά Συνδεδεμένες Λίστες (συν.)

Βοηθητική συνάρτηση: Δημιουργία κόμβου

```
NODE* createListNode(type x){
```




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

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;
  struct node *previous;
} NODE;

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

Συναρτήσεις που πρέπει να υλοποιήσετε

- `NODE* createListNode(type x);`
- `void makeEmptyList(DLIST *L);`
- `bool isEmptyList(DLIST *L);`
- `void insertFirstNode(DLIST *L,type x);`
- `void printList(DLIST *L);`
- `void printListReverse(DLIST *L);`
- `void deleteFirstNode(DLIST *L);`
- `void deleteLastNode(DLIST *L);`
- `void swapFirstWithLast(DLIST *L);`

Λύσεις

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

Λύσεις (συν.)

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

Λύσεις (συν.)

```
bool IsEmptyList(DLIST *L) {
   return (L->size == 0);
}
```

Λύσεις (συν.)

```
void insertFirstNode(DLIST *L, type x){
  NODE *newNode = createListNode(x);
  if( !isEmptyList(L) )
    L->head->previous = newNode;
  newNode->next = L->head;
  L->head = newNode;
  L->size++;
}
```

Λύσεις (συν.)

```
void printList(DLIST *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 printListReverse(DLIST *L) {
  NODE* tmp = L->head;
  for(int i=0; i<(L->size-1); i++){
    tmp = tmp->next;
  }
  while(tmp!=NULL){
    printf("%d ", tmp->data);
    tmp = tmp->previous;
  }
  printf("\n");
}
```

Λύσεις (συν.)

```
void deleteFirstNode(DLIST *L){
  NODE *tmp;
  if( !isEmptyList(L) ) {
    if(L->size==1) {
      free(L->head);
      L->head = NULL;
    }
    else{
      tmp = L->head;
      L->head = L->head->next;
      L->head->previous = NULL;
      free(tmp);
    }
    L->size--;
  }
}
```

Λύσεις (συν.)

```
void deleteLastNode(DLIST *L){
  NODE *tmp;
  if( !isEmptyList(L) ) {
    if(L->size==1) {
      free(L->head);
      L->head = NULL;
    }
    else{
      tmp = L->head;
      for(int i=0; i<(L->size-1); i++){
        tmp = tmp->next;
      }
      tmp->previous->next = NULL;
      free(tmp);
    }
    L->size--;
  }
}
```

Λύσεις (συν.)

```
void swapFirstWithLast(DLIST *L) {
  NODE *tmpNode = L->head;
  type tmpData;
  if ( !isEmptyList(L) &&  L->size>1) {
    for (int i=0; i < (L->size-1); i++)
      tmpNode = tmpNode->next;

    //swap head data with last node data
    tmpData = tmpNode->data;
    tmpNode->data = L->head->data;
    L->head->data = tmpData;
  }
}
```