Τμήμα Πληροφορικής
Πανεπιστήμιο Κύπρου

ΕΠΛ232 –Προγραμματιστικές Τεχνικές και Εργαλεία

Διάλεξη 15: Δομές Δεδομένων IV (Διπλά Συνδεδεμένες Λίστες)

Δημήτρης Ζεϊναλιπούρ

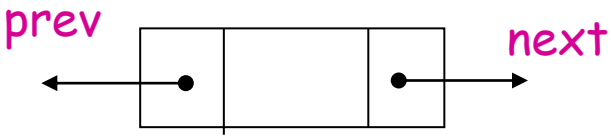
http://www.cs.ucy.ac.cy/courses/EPL232

Περιεχόμενο Διάλεξης 15

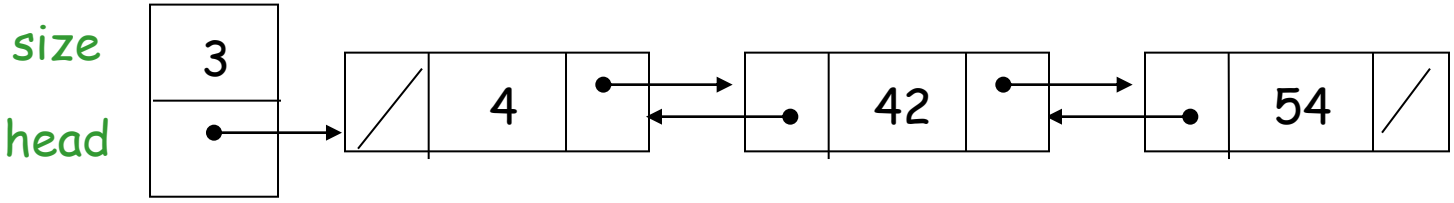
- **Διπλά Συνδεδεμένες Λίστες**
  - Ορισμοί και Δηλώσεις
  - Υλοποίηση Συνάρτησης `put(l, x, y)` **αναδρομικά** και **επαναληπτικά**
- **Ταξινομημένες Διπλά Συνδεδεμένες Λίστες**
  - Υλοποίηση Συνάρτησης `printlist(l)`
  - Υλοποίηση Συνάρτησης `insert(l, x)`
  - Υλοποίηση Συνάρτησης `delete(l, x)`

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:

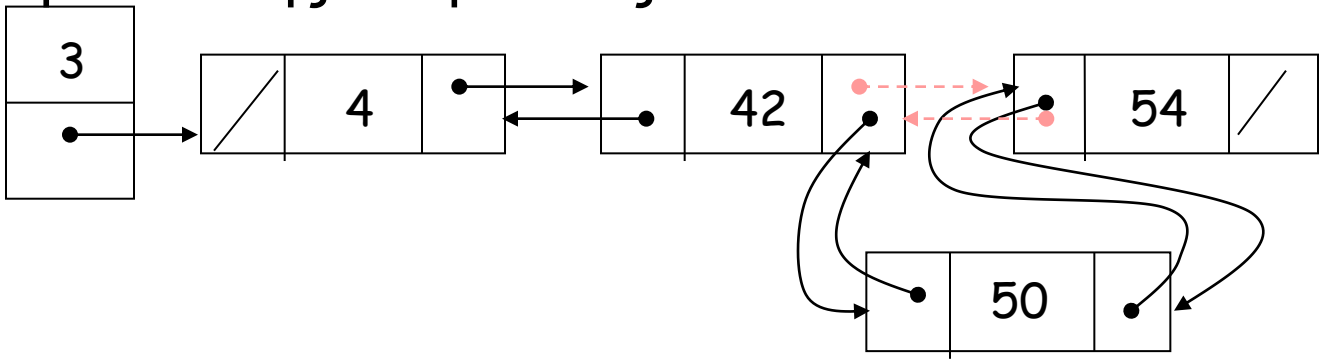
```
typedef struct dnode {
        int       data;
        struct dnode *prev;
        struct dnode *next;
} DNODE;
```

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

```
typedef struct {
        DNODE *head;
        int   size;
} DLLIST;
```

Διπλά Συνδεδεμένες Λίστες

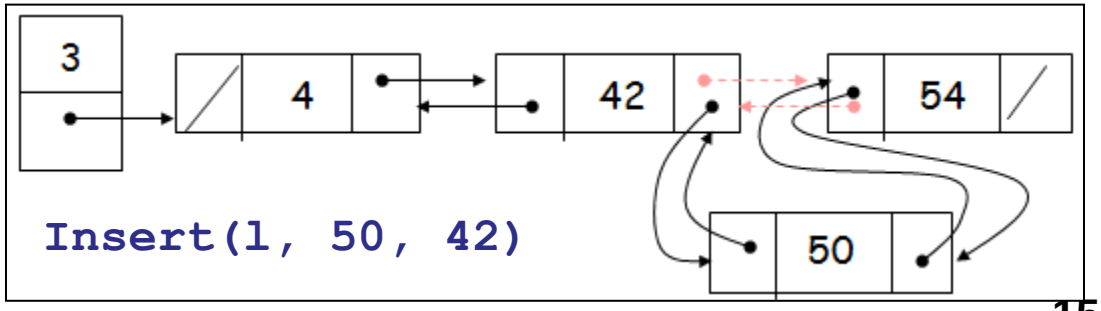
- Προφανώς η **εισαγωγή** στοιχείου σε **κάποιο σημείο** μιας διπλά συνδεδεμένης λίστας περιέχει κάποια **επιπλέον πολυπλοκότητα** από την εισαγωγή σε μια **μονά-συνδεδεμένη** λίστα.
- Αυτό γιατί κάθε **νέος κόμβος** πρέπει να **συνδεθεί** και με τον **επόμενο** και με τον **προηγούμενο** κόμβο στη λίστα. Παρόμοια, κατά τις εξαγωγές στοιχείων.
- **Παράδειγμα εισαγωγής** του στοιχείου **50** μετά το **42** στη λίστα της διαφάνειας **17**:



Η συνάρτηση put

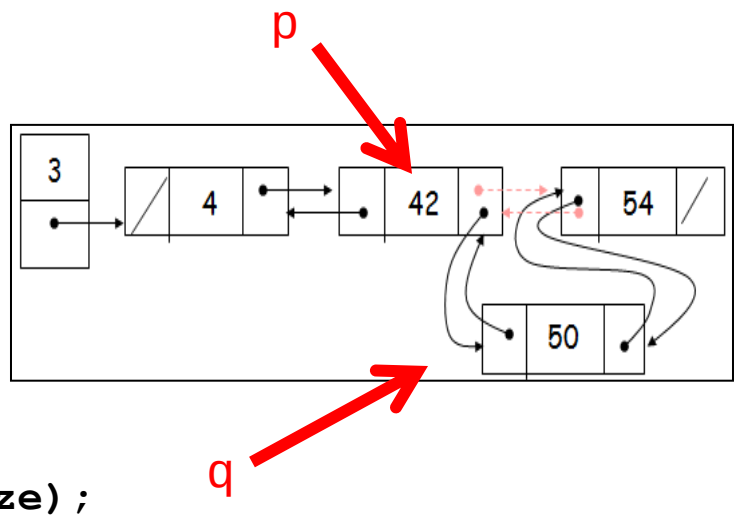
- Να ορίσετε συνάρτηση **put(l, x, y)** η οποία τοποθετεί το στοιχείο **x** μετά από το στοιχείο **y** μέσα στη λίστα **l**, αν το στοιχείο **y** υπάρχει στην λίστα.

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



Η συνάρτηση `puttnode` (Αναδρομική)

```
static int putnode(DLNODE *p, int x, int y, int *size){
  DLNODE  *q = NULL;
  if (p == NULL) {
        printf("Element %d does not exist, y); return EXIT_FAILURE;
  }
  else if ( p->data == y ) {
        q = (DLNODE *)malloc(sizeof(DLNODE));
        if (q == NULL) {
           printf("Error: Malloc"); return EXIT_FAILURE;
        }
        q->data = x;  q->next = p->next;
        q->prev = p;
        if (p->next != NULL)
           (p->next)->prev = q;
        p->next = q;    (*size)++
        return EXIT_SUCCESS;
  }
  else
        return putnode(p->next, x, y, size);
  }
}
```



Ταξινομημένες Διπλά Συνδεδεμένες Λίστες

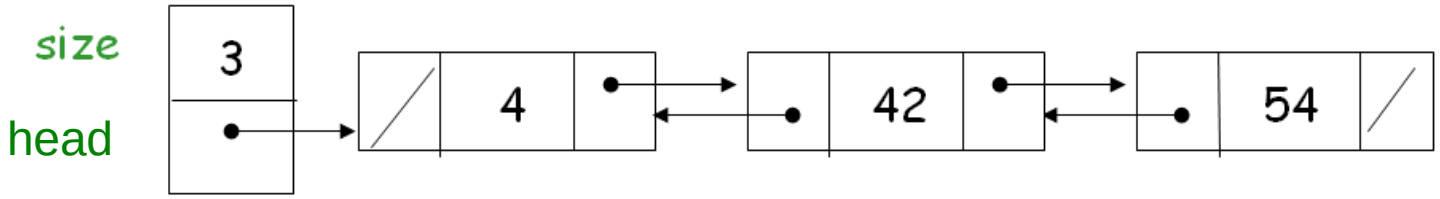
Άσκηση: Να ορίσετε τις **αναδρομικές** διαδικασίες:

- `void printlist(DLLIST *l)`, η οποία τυπώνει όλα τα στοιχεία της λίστας l.
- `void insert(DLLIST *l, int x)`, η οποία εισάγει το στοιχείο x μέσα στη λίστα l,
- `void delete(DLLIST *l, int x)`, η οποία αφαιρεί το στοιχείο x από τη λίστα l (αν αυτό υπάρχει) και

- Όλες οι πιο πάνω συναρτήσεις πρέπει να διατηρούν *ταξινομημένες* τις διπλά συνδεδεμένες λίστες.

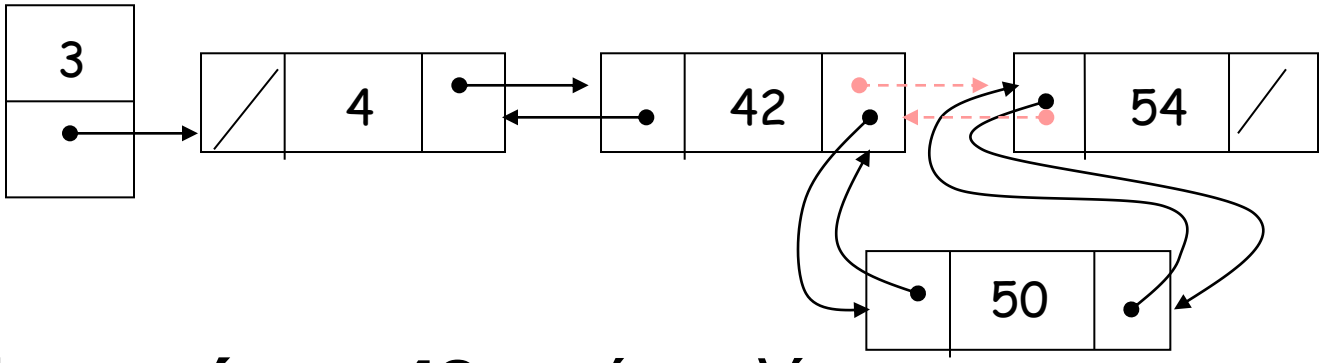
Η Συνάρτηση `printList`

```
void printlist(DLLIST *l){
   if (l == NULL || l->size == 0)
        printf("The list is empty\n");
   else
        printnode(l->head);
}
void printnode(DLNODE *p){
   if (p != NULL){
        printf("%d", p->data);
        printnode(p->next);
    }
}
```

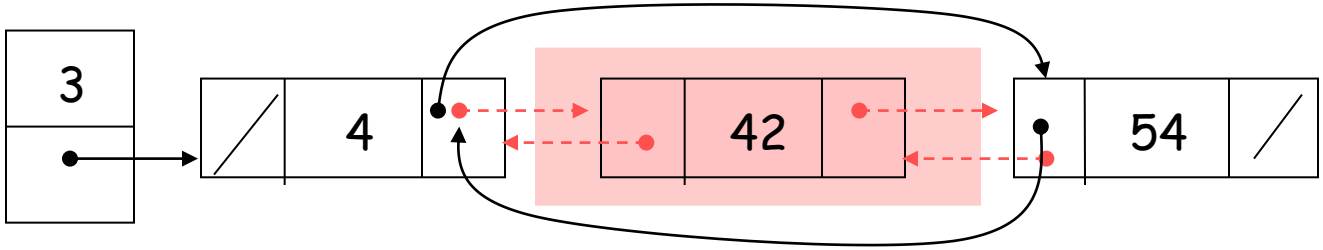


Παραδείγματα

- **Εισαγωγή** του **50** στη λίστα:



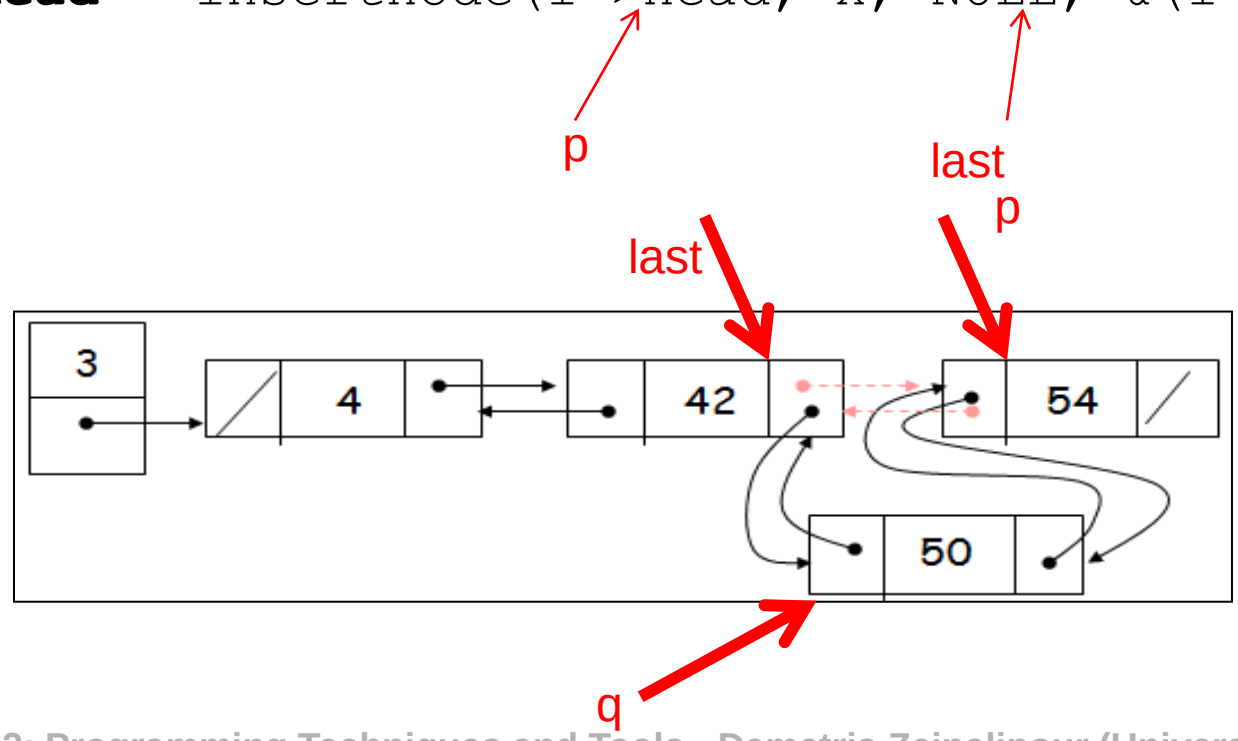
- **Εξαγωγή** του **42** από τη λίστα:



Θα δούμε κάποιους διαφορετικούς τρόπους υλοποίησης.
Επιλέξτε τον τρόπο που σας βολεύει καλύτερα.

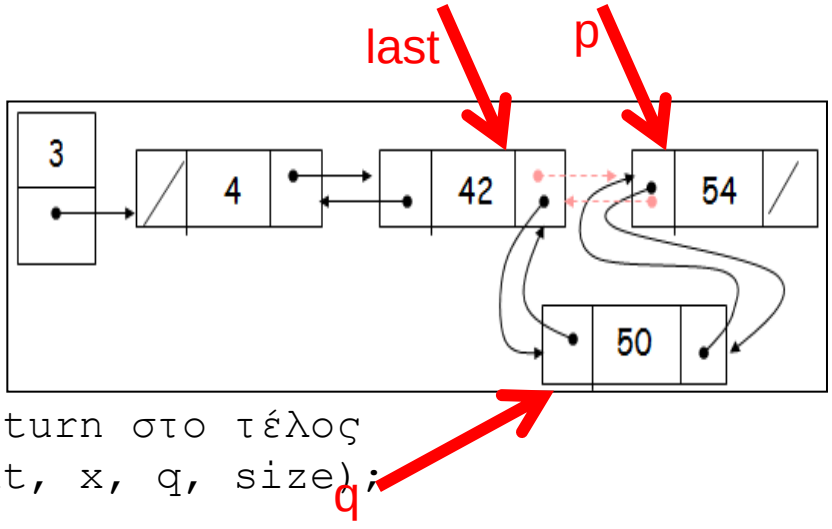
Η Συνάρτηση `insert`

```
void insert(DLLIST *l, int x) {
 if (l == NULL) {
      printf("list is empty\n"); return;
 }
 l->head = insertnode(l->head, x, NULL, &(l->size));
}
```



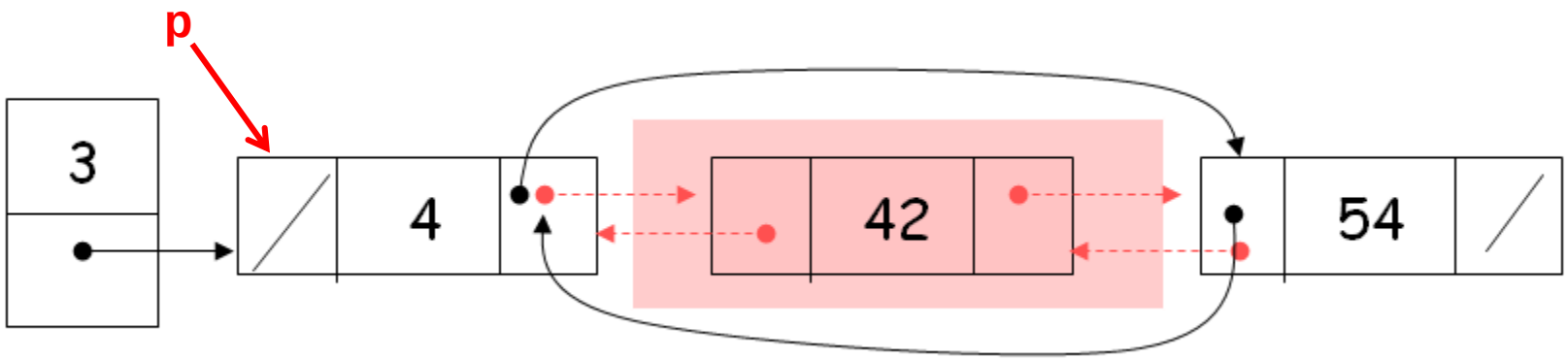
Η Συνάρτηση insert

```
DLNODE *insertnode(DLNODE *p,  int x, DLNODE *last, int *size) {
   DLNODE *q = NULL;
   if (p == NULL){// Φτάσαμε στο τέλος της λίστας
        q = (DLNODE *) malloc(sizeof(DLNODE));
        q->data = x; q->next = NULL;
        q->prev = last; (*size)++;
   }
   else if (p->data >= x) { // Περάσαμε το σημείο εισαγωγής
        q = (DLNODE *) malloc (sizeof(DLNODE));
        q->data = x;
        q->next = p;
        q->prev = p->prev;
        if ((p->prev) != NULL)
            (p->prev)->next = q;
        p->prev = q; (*size)++;
   }
   else {
        q = p; // για ομοιόμορφο return στο τέλος
        q->next = insertnode(q->next, x, q, size);
   }
   return q; // για όλους τους κόμβους (εκτός του κόμβου
   εισαγωγής) αυτό είναι το p ουσιαστικά
}
```



Η Συνάρτηση `delete`

```
void delete(DLLIST *l, int x){
 l->head = deletenode(l->head, x, &(l->size));
}
```



Η Συνάρτηση delete

```
DLNODE *deletenode(DLNODE *p, int x, int *size){
  DLNODE  *q = NULL;
  if ( (p == NULL) || (p->data > x) ){
        printf("Item %d not found \n", x);
   }
  else if ( p->data == x ){
       q = p->next;
       if (q != NULL)
             q->prev = p->prev;
       if ((p->prev)!= NULL)
              (p->prev)->next = q;
       free(p);
       printf("Item %d has been deleted\n", x);
       (*size)--;
       p = q; // θα γίνει return το p.
  }
  else {
       p->next = deletenode(p->next, x, size);
   }
  return p; // απαιτείται για επιβεβαίωση του backlink σε διαγραφές
}
```