Διάλεξη 10: 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: Δημήτρης Ζεϊναλιπούρ

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)**

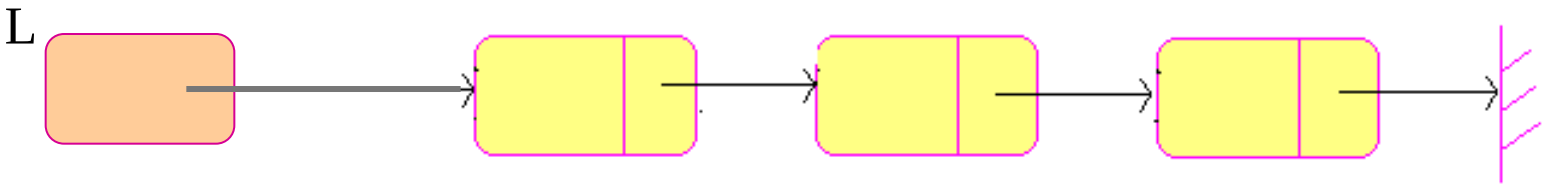
- Κάποιες βασικές πράξεις των λιστών είναι οι πιο κάτω:

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:

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

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

Ευθύγραμμες Απλά Συνδεδεμένες Λίστες

- Πιο κάτω ορίζονται κάποιες χρήσιμες πράξεις.
- *Εύρεση Κόμβου με συγκεκριμένο στοιχείο*:

```
int find(LIST *L, key val){
    p = L->head;
    while (p != NULL) {
       if (p->data == val)
          return 0;
       p = p->next;
    }
    return 1;
}
```

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

Ευθύγραμμες Απλά Συνδεδεμένες Λίστες

- Με παρόμοιο τρόπο μπορούμε να ορίσουμε διαδικασία findpointer(LIST *L, key val) που επιστρέφει δείκτη προς κόμβο της λίστας που περιέχει το στοιχείο val, αν υπάρχει.

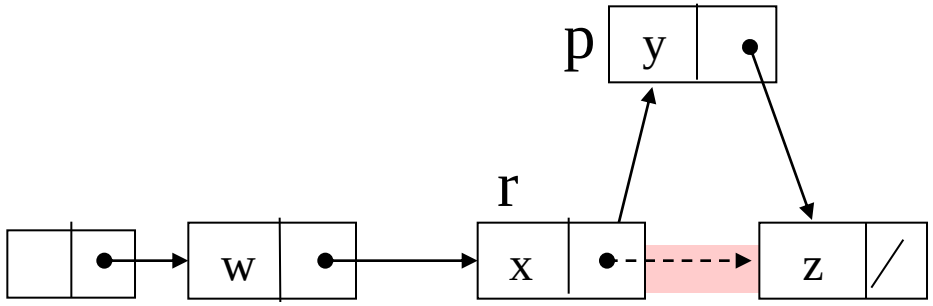
```
NODE *findpointer(LIST *list, int value) {

  NODE *p = list->head;

  while (p != NULL) {
   if (p->val == value) {
          return p;
   }
   p = p->next;
  }

  return NULL;
}
```

Πως θα το υλοποιήσετε αναδρομικά;

Ευθύγραμμες Απλά Συνδεδεμένες Λίστες

- *Εισαγωγή Κόμβου "x" μετά από συγκεκριμένο κόμβο "y"*

```
int insert(LIST *L, value x, value y) {
    NODE *r = findpointer(L,x);
    if r == NULL return 1;
    else {
        p = (NODE *)malloc(sizeof(NODE));
        p->data = y;
        p->next = r->next;
        r->next = p;
    }
    return 0;
}
```



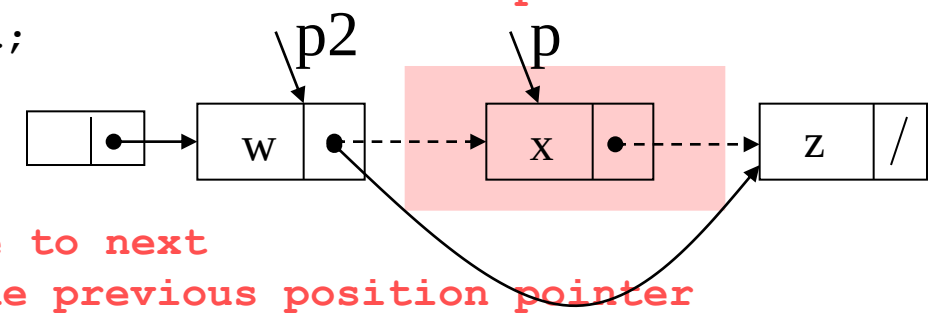
Ευθύγραμμες Απλά Συνδεδεμένες Λίστες

- Εξαγωγή Κόμβου με συγκεκριμένη πληροφορία "x"

```
typedef struct {
      type  val;
      struct node *next;
} NODE;
```

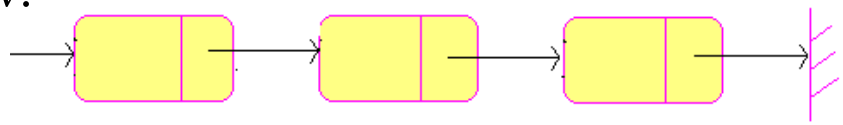
```
typedef struct list {
   NODE  *head;
   int size;
} LIST;
```

```
int delete(LIST *L, key x){
   NODE *p = NULL, *p2 = NULL;
   p = list->head;
   if (p == NULL)  return 1;

   if (p->val == value) {// the head is treated separately
        list->head = p->next; list->size--; free(p); return 0;
   }
   else {
        while (p != NULL) {// find x and delete it
          if (p->val == x) {// value found => delete p
               p2->next = p->next;
               free(p);
               list->size--;
               return 0;
          } // x not found – move to next
          p2 = p; // use p2 as the previous position pointer
          p = p->next;
   }
   return 1;
}
```



Πίνακες vs. Συνδεδεμένες Λίστες

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



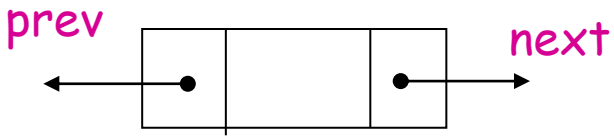
- **Χρήση Χώρου**
  - **Πίνακας:** καταλαμβάνεται ο ίδιος χώρος άσχετα με τον αριθμό των στοιχείων που είναι αποθηκευμένα.
  - **Συνδεδεμένη Λίστα:** μέγεθος της λίστας × (χώρος ενός στοιχείου + χώρος ενός δείκτη)

- **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

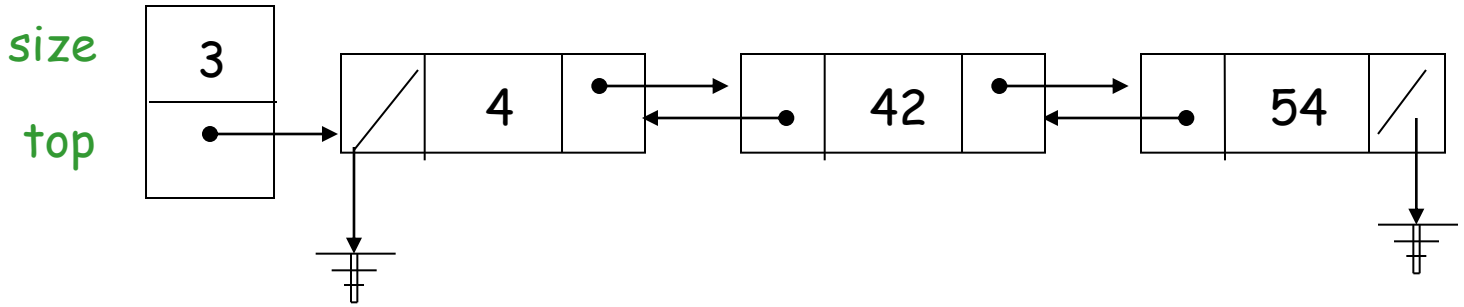
Ευθύγραμμες Διπλά Συνδεδεμένες Λίστες

- **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:

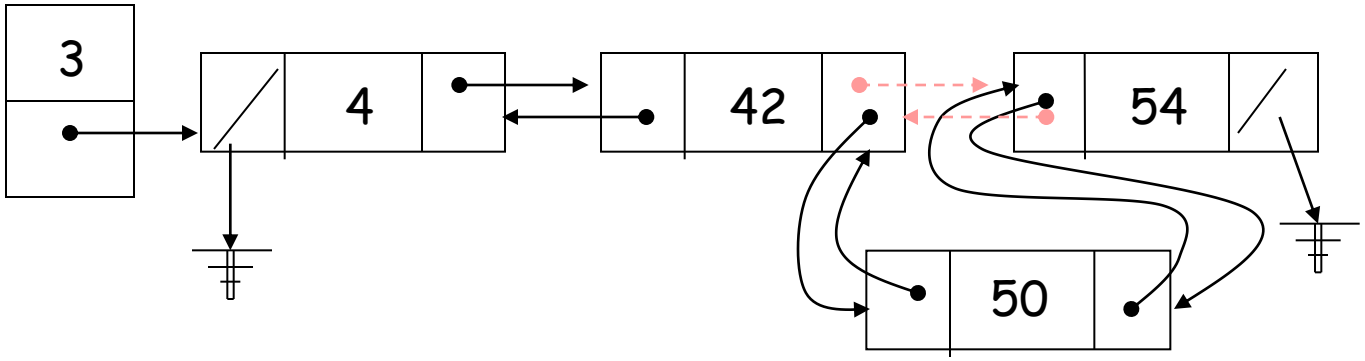
```
typedef struct dlnode {
        int             data;
        struct dlnode *prev;
        struct dlnode *next;
} DLNODE;
```

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

```
typedef struct {
    DLNODE *head;
    ...
} 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 50 μετά το 42 στη λίστα της διαφάνειας 8:



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);
  }
}
```