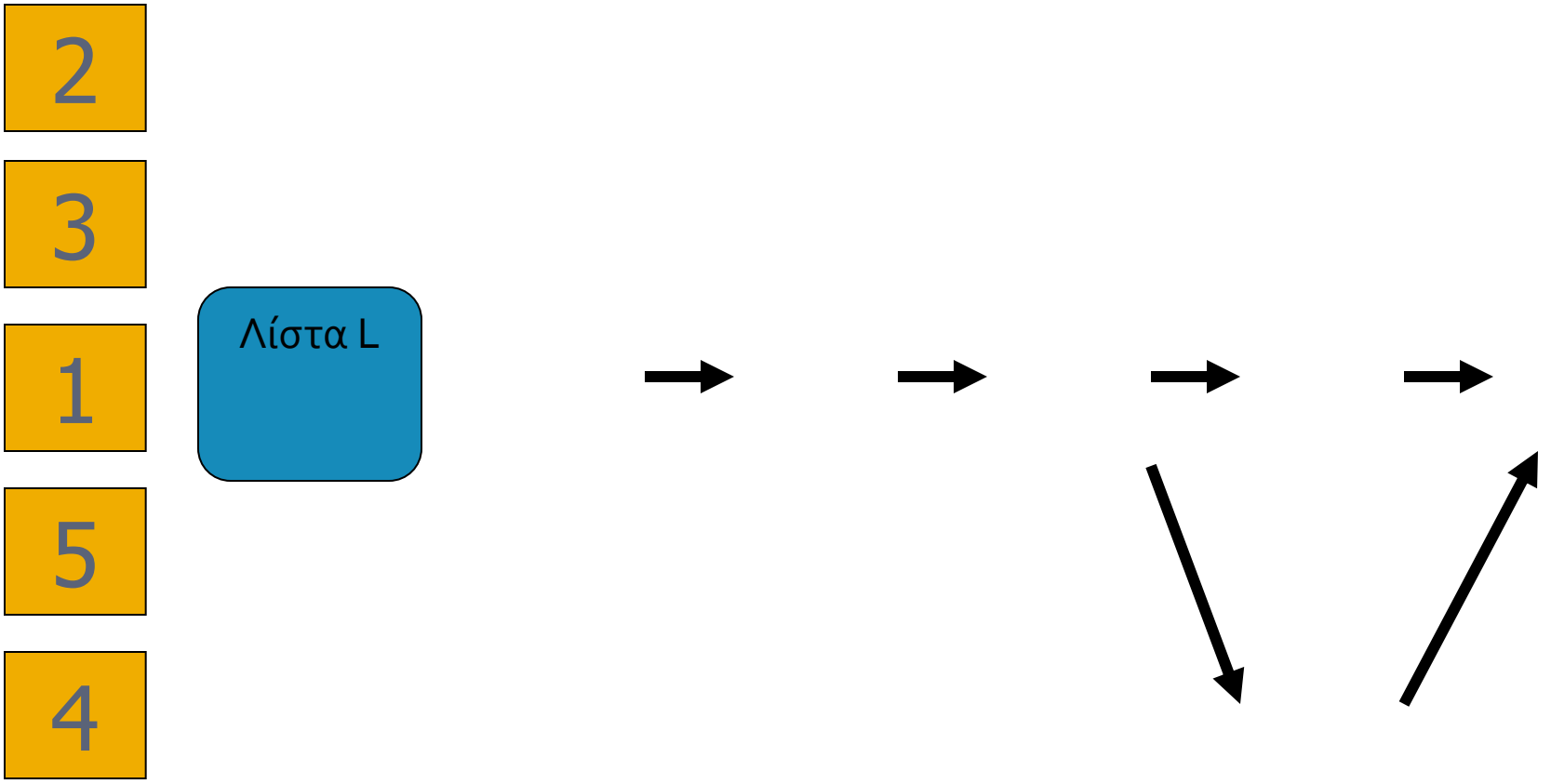
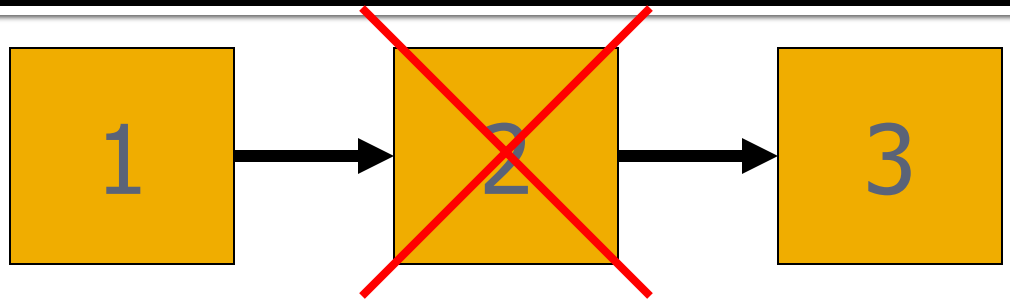
Lab 3: Sorted List

EPL231 – Data Structures and Algorithms

Ταξινομημένη λίστα



Think about DELETE



1. **Ο απλός τρόπος**

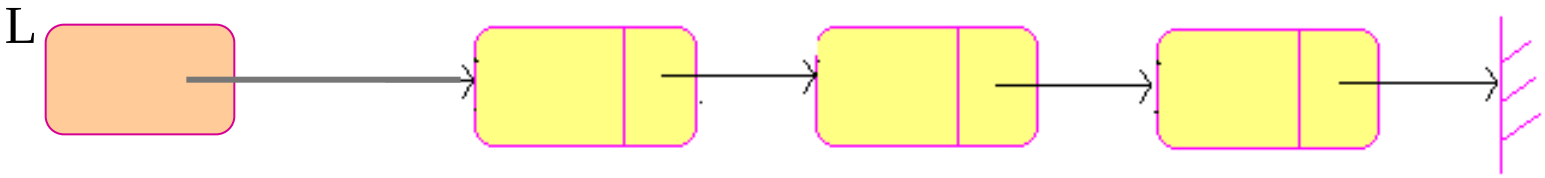
   `node1→next = node2→next`

   - Πρόβλημα?
     - Ο κόμβος Node2 παραμένει στην μνήμη
2. **Ο ΣΩΣΤΟΣ τρόπος (απελευθέρωση αχρείαστης μνήμης)**

```
tempNode = node2
node1→next = node2→next
free(tempNode)
```

Τι μάθαμε στις διαλέξεις

- Ευθύγραμμες Απλά Συνδεδεμένες Λίστες
  - Μία ταξινομημένη λίστα μπορεί να υλοποιηθεί ως μια συνδεδεμένη λίστα (με παρόμοιο τρόπο όπως μια στοίβα).



Δηλώσεις Τύπων

```
// Declare the data type
typedef struct _ListNode {
     Type              data;
     struct _ListNode *next;
} ListNode;

// Declare a pointer to a List Node.
typedef ListNode *ListNodePtr;

// Declare a List
typedef struct _List {
     int size;
     ListPtr firstNode;
} List;

// Declare a List Pointer
typedef List *ListPtr;
```

Ευθύγραμμες Απλά Συνδεδεμένες Λίστες

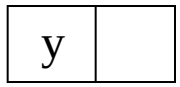
- Πιο κάτω ορίζονται κάποιες χρήσιμες πράξεις.
- *Εύρεση Κόμβου με συγκεκριμένο στοιχείο*:

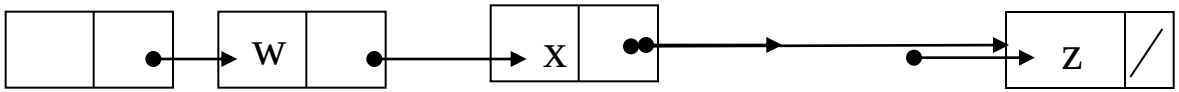
```
bool find(List *L, Type val){
    return findnode(L->top, val);
}

bool findnode(ListNode *P, Type val){
for ( Q = P; Q != NULL; Q = Q->next);
    if Q->data == val)
            return TRUE;
return FALSE;
}
```

Ευθύγραμμες Απλά Συνδεδεμένες Λίστες

- Με παρόμοιο τρόπο μπορούμε να ορίσουμε διαδικασία
  - **`findpointer`**`(List *L, Type val)`
  - Επιστρέφει δείκτη προς κόμβο της λίστας που περιέχει το στοιχείο val, αν υπάρχει.
- *Εισαγωγή Κόμβου μετά από συγκεκριμένο κόμβο*

  `insert``(List *L, Type x,y)`



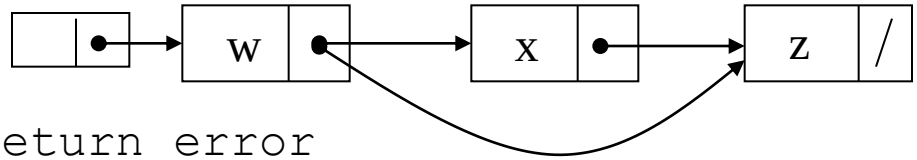


Ευθύγραμμες Απλά Συνδεδεμένες Λίστες

```
insert(List *L, Type x,y) {
    r = findpointer(L,x);
    if (r == NULL)
      error
    else {
      q = (ListNode *)malloc(sizeof(NODE));
      q->data = y;
       q->next = r->next;
      r->next = q;
     }
}
```

Ευθύγραμμες Απλά Συνδεδεμένες Λίστες

- Εξαγωγή Κόμβου με συγκεκριμένη πληροφορία




```
delete(List *L, key x){
     if (L->top == NULL) return error
     Q = L->top;
     if Q->data == x
         L->top = Q->next; free(Q);
     else {
         Found = False; R = Q; Q = Q->next;
          while(Q != NULL & Found==False){
              if (Q->data == x) Found = True ;
              else R = Q; Q = Q->next;
       }
     if Q == null  error
     else R->next = Q->next; free(Q);
}
```

Συναρτήσεις για υλοποίηση

- **IsEmpty(L)** επέστρεψε true αν η λίστα L είναι κενή
- **PrintList(L)** τυπώνει τη ταξινομημένη λίστα L
- **Insert(x, L)** εισήγαγε το x μέσα στη ταξινομημένη λίστα L διατηρώντας την L ταξινομημένη
- **Delete(x, L)** αφαίρεσε το x από την ταξινομημένη λίστα L
- **Access(L, i)** επέστρεψε το i-οστό στοιχείο της L
- **DeleteMin(L)** επέστρεψε το μικρότερο στοιχείο της λίστας L και αφαίρεσε το από τη λίστα
- **DeleteMax(L)** επέστρεψε το μέγιστο στοιχείο της λίστας L και αφαίρεσε το από τη λίστα

Συναρτήσεις

- Υλοποιήστε τα ακόλουθα:
  - BOOL **isEmpty**(SortedList *List): TRUE, αν η λίστα είναι κενή.
  - void **initList**(SortedList *List): Δημιουργία κενής λίστας.
  - void **printList**(SortedList *List): Τύπωσε τα περιεχόμενα της λίστας.
  - void **insertData**(SortedList *List, Type data): Εισήγαγε δεδομένα στη λίστα, φροντίζοντας η λίστα να παραμείνει ταξινομημένη.
  - BOOL **deleteData**(SortedList *List, Type data): Διάγραψε δεδομένα από τη λίστα.

Συναρτήσεις

- Υλοποιημένα:
  - ListNode *__accessNode__(SortedList *List, int index): Επέστρεψε τον κόμβο στη θέση index.
  - BOOL __deleteMin__(SortedList *List): Διέγραψε το στοιχείο με την ελάχιστη τιμή.
  - BOOL __deleteMax__(SortedList *List): Διέγραψε το στοιχείο με την μέγιστη τιμή.
  - void __clearList__(SortedList *List): Διαγραφή των περιεχομένων της λίστας.
  - ListNode *__findNode__(SortedList *List, Type key): Βρές τον κόμβο με συγκεκριμένη τιμή.
  - BOOL __exists__(SortedList *List, Type key): Ελέγχει αν υπαρχει στοιχείο με τιμή key.

The End