

ΕΠΛ232 – Προγραμματιστικές Τεχνικές και Εργαλεία

Δεδομένων III (Λίστες και Παραδείγματα)

Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου

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

**Το μάθημα αυτό δομήθηκε βάση των διαλέξεων του
Αναπλ. Καθηγητή Δημήτρη Ζεϊναλιπούρ**



Περιεχόμενο Διάλεξης 14

- **Υλοποίηση Insert/Delete σε Λίστα**

- Επαναληπτική & Αναδρομική Λύση

- **Προβλήματα Λιστών**

1. Απάλειψη Διπλοτύπων από Λίστα – `eliminateDuplicates(list)`
 - Επαναληπτική & Αναδρομική Λύση
2. Συγχώνευση Ταξινομημένων Ακολουθιών – `mergeLists(list1, list2, list3)`
 - Επαναληπτική & Αναδρομική Λύση
3. Βελτιωμένες Δομές Δεδομένων Λύσης

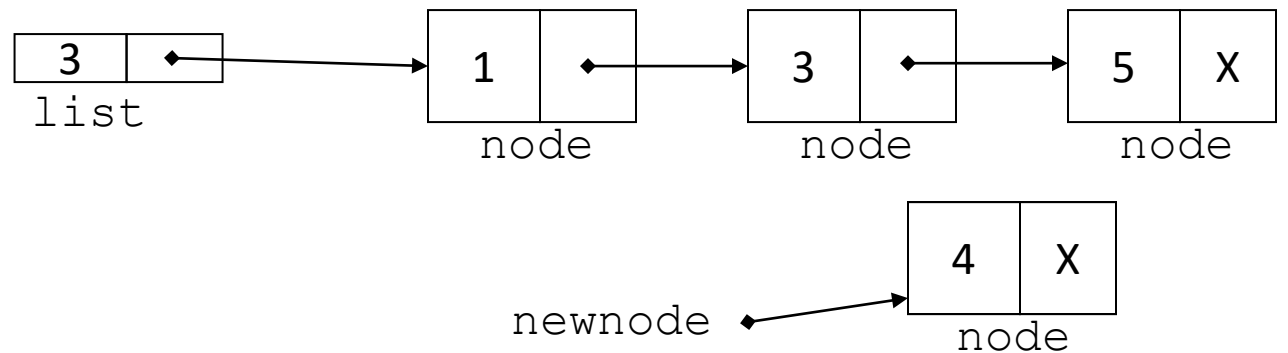


Ταξινομημένες Λίστες

Συνάρτηση `insert`

```
void insert(LIST *l, int x){
    NODE *p = NULL, *q = NULL; /* prev, curr copy pointers*/

    if (l == NULL) { printf("Unable to Enter"); return; }
    /* Create and Fill New Node */
    NODE * newNode = (NODE *)malloc(sizeof(NODE));
    if (newNode == NULL) {
        printf("Error: Unable to Allocate Memory!");
        return;
    }
    newNode->data = x;
    newNode->next = NULL;
```



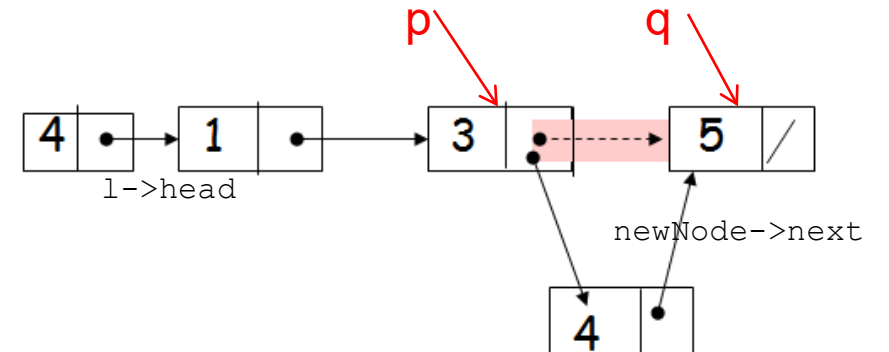
συνέχεια...



Ταξινομημένες Λίστες

Συνάρτηση `insert`

```
/* 2) Connect newNode to List */
if ( l->head == NULL ) /* If Empty add to l->head */
    l->head = newNode;
else {
    p = q = l->head; /* Fast forward p and q until data<x*/
    while ( ( q != NULL ) && ( q->data < x ) ) {
        p = q;
        q = q->next;
    }
    if ( p == q ) { /* i.e., Insert to first position */
        newNode->next = l->head; // e.g., insert "0"
        l->head = newNode; // Branch required for this!
    } else { /* Insert anywhere else */
        newNode->next = q;
        p->next = newNode;
    }
} // else()
(l->size)++; /* Increase List size*/
} // insert()
```



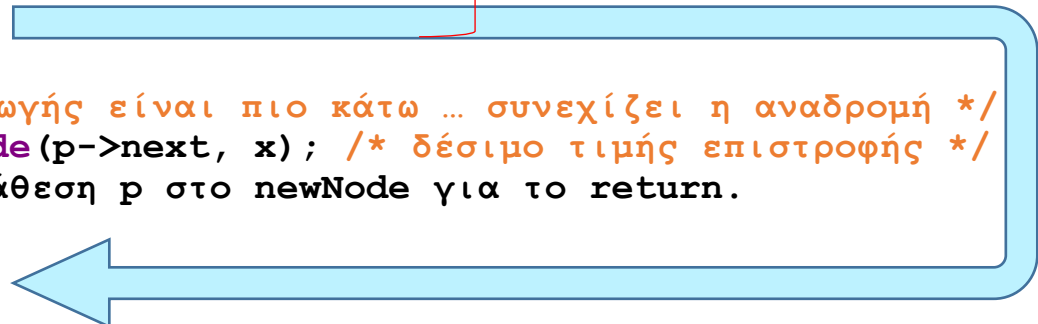
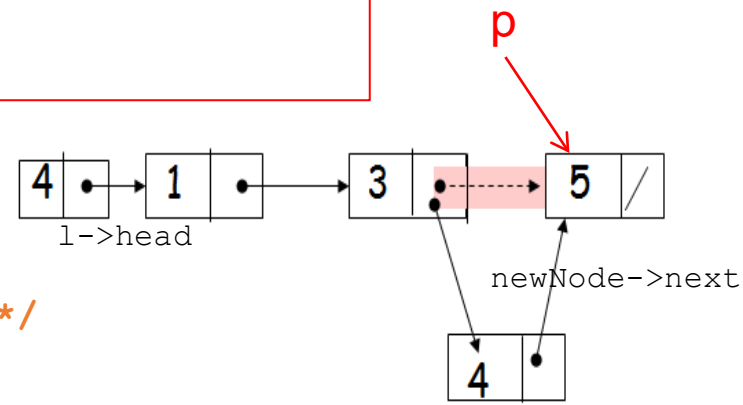
Ταξινομημένες Λίστες

Αναδρομική Συνάρτηση `insert`

```
void insert(LIST *l, int x){
    ++(l->size); // η προκαταβολική αύξηση θα βελτιωθεί σε λίγο
    l->head = insertnode(l->head, x);
}
```

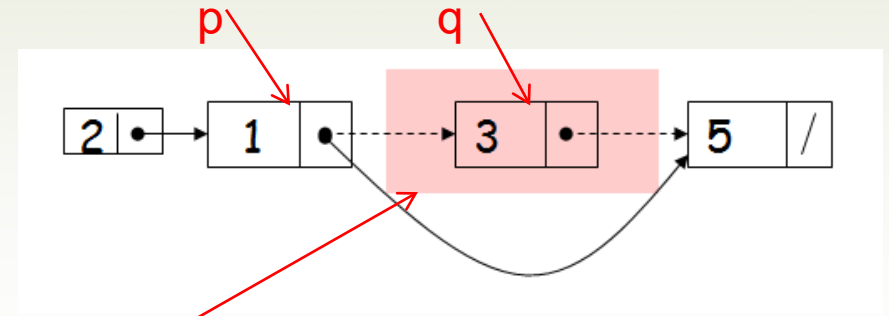
```
static NODE *insertnode(NODE *p, int x){
    NODE *newNode = NULL;
    if (p == NULL){ /* Η λίστα είναι άδεια */
        newNode = (NODE *)malloc(sizeof(NODE));
        newNode->data = x;
        newNode->next = NULL;
    }
    else if ( x <= p->data ) { /* Το Σημείο Εισαγωγής μόλις ξεπεράστηκε */
        newNode = (NODE *)malloc(sizeof(NODE));
        newNode->data = x;
        newNode->next = p;
    }
    else { /* Το σημείο εισαγωγής είναι πιο κάτω ... συνεχίζει η αναδρομή */
        p->next = insertnode(p->next, x); /* δέσιμο τιμής επιστροφής */
        newNode = p; // ανάθεση p στο newNode για το return.
    }
    return newNode;
}
```

```
if ((p == NULL) || (p->data >= x)) {
    newNode = (NODE *)malloc(sizeof(NODE));
    newNode->data = x;
    newNode->next = p;
}
```



Ταξινομημένες Λίστες

Συνάρτηση `delete`



```
void delete(LIST *l, int x){
    NODE *p = NULL, *q = NULL;
    if ((l==NULL) || (l->head==NULL)) { printf("Error"); return; }
    p = q = l->head;
    // Fast Forward (ffwd) list
    while ( ( q != NULL ) && ( q->data < x ) ) {
        p = q;           // p: prev
        q = q->next;    // q: next
    } // end of list    // just passed the possible position for x
    if ( ( q == NULL ) || ( q->data > x ) )
        printf("Item %d not found \n", x);
        return;
    }
    if (p == q) // Item to be deleted is in 1st position
        l->head = l->head->next; // initially p, q point to 1st
    else // ffwd took us somewhere else in the list.
        p->next = q->next; // fix pointer of prev
    free( q ); // deallocate curr pointer
    (l->size)--; // adjust list size
}
```



Αναδρομή ME return στην οπισθοχώρηση και τιμή δια αναφοράς

Ταξινομημένες Λίστες

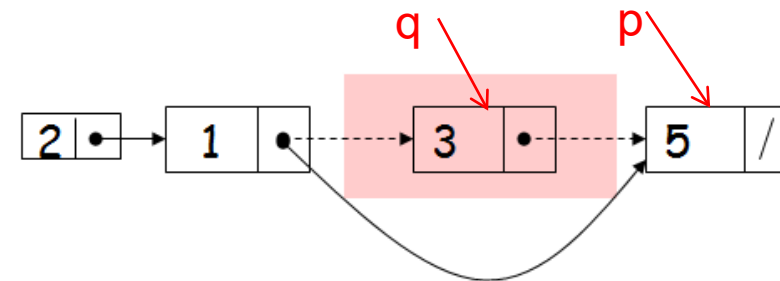
Αναδρομική Συνάρτηση `delete`

```
void delete (LIST *l, int x){  
    if ((l==NULL) || (l->head==NULL)) { printf("Error"); return; }  
    l->head = deletenode(l->head, x, (&(l->size)));  
}
```

Καλύτερο από προσέγγιση με το `insert`

```
static NODE *deletenode (NODE *p, int x, int *size){  
    if ( ( p == NULL) || (p->data > x) ) {  
        printf("Item %d not found \n", x);  
    }  
    else if ( p->data == x ){//item found  
        NODE *q = NULL;  
        q = p;  
        p = p->next; // to be returned  
        free(q);  
        (*size)--;  
    } else {  
        p->next = deletenode(p->next, x, size);  
    }  
    return p;  
}
```

Surpassed possible position of x or reached end of list



Πρόβλημα 1: `eliminateDuplicates`

- Γράψετε συνάρτηση στη γλώσσα C η οποία παίρνει ως όρισμα ένα **δείκτη σε ταξινομημένη λίστα** και η οποία **απαλείφει** τα όποια **διπλότυπα** ενδέχεται να υπάρχουν.

- **Πρότυπο Συνάρτησης:**

```
int eliminateDuplicates(LIST *list);
```

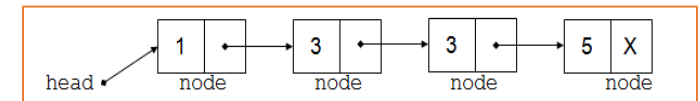
```
typedef struct node {  
    int data;  
    struct node *next;  
} NODE;
```

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

- **Κλήση Συνάρτησης**

```
LIST *list; initList2(&list); fillList(list);  
    eliminateDuplicates(list); printlist(list);
```

→ τυπώνει 1, 3, 5



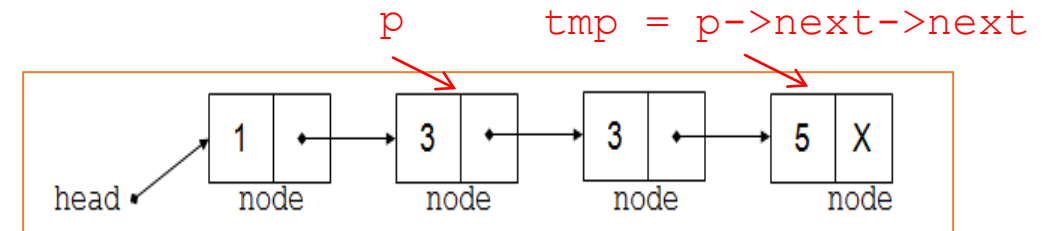
Πρόβλημα 1: `eliminateDuplicates`

Επαναληπτική Λύση

```
static void eliminateDuplicateList(NODE *p, int *size) {
    if (p == NULL)
        return;           // do nothing if the list is empty

    while(p->next!=NULL) {
        if (p->data == p->next->data) { // next is a duplicate
            NODE *tmp = p->next->next;
            free(p->next);
            p->next = tmp;
            (*size)--;
        }
        else
            p = p->next;
    }
}

int eliminateDuplicates(LIST *l) {
    if (l == NULL) return EXIT_FAILURE;
    eliminateDuplicateList(l->head, &(l->size));
    return EXIT_SUCCESS;
}
```

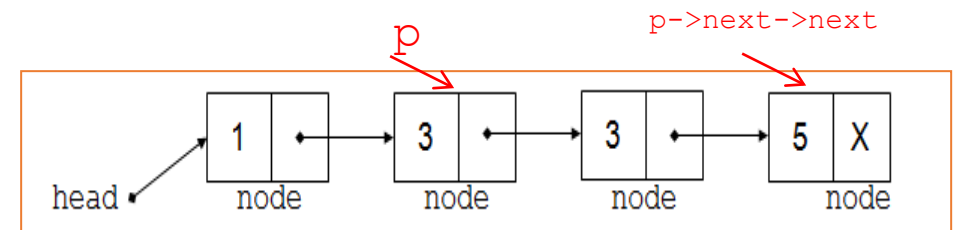


Πρόβλημα 1: `eliminateDuplicates`

Αναδρομική Λύση

```
static void eliminateDuplicateRecursive(NODE *p, int *size) {
    if (p == NULL) return; // do nothing if the list is empty
    if (p->next != NULL) {
        if (p->data == p->next->data) {
            NODE *tmp = p->next->next;
            free(p->next);
            p->next = tmp;
            (*size)--;
        }
        else
            p = p->next;
        eliminateDuplicateRecursive(p, size);
    }
}

int eliminateDuplicates(LIST *l) {
    if (l == NULL) return EXIT_FAILURE;
    eliminateDuplicateList(l->head, &(l->size));
    return EXIT_SUCCESS;
}
```



Κατά τα άλλα πολύ όμοια
με την επαναληπτική
έκδοση



Πρόβλημα 2: mergeLists

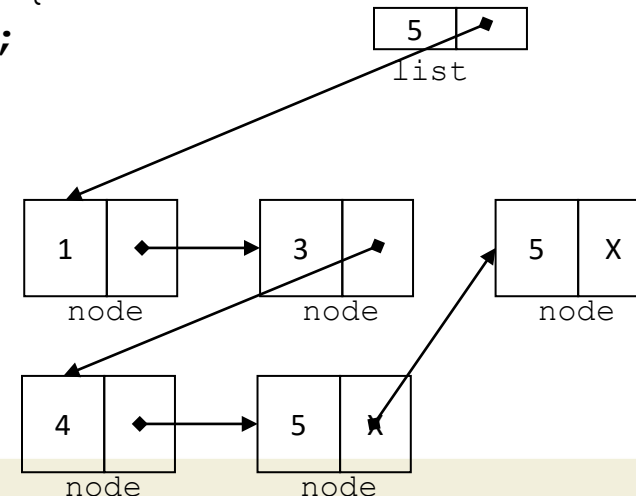
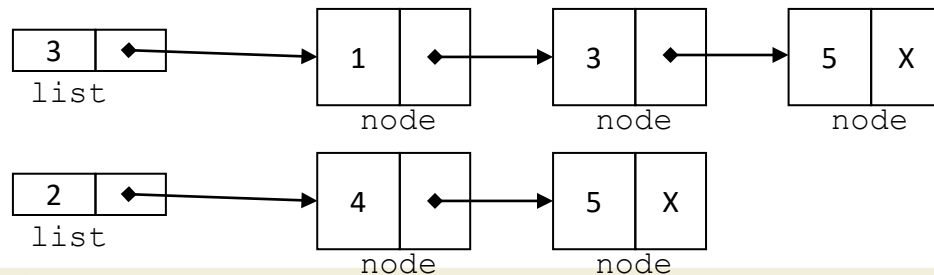
- Γράψετε συνάρτηση στη C η οποία παίρνει ως όρισμα δυο δείκτες σε ταξινομημένες λίστες και η οποία συγχωνεύει (ταξινομημένα) τις δυο λίστες χωρίς δημιουργία ενδιάμεσης λίστας.

- Πρότυπο Συνάρτησης:

```
int mergeLists(LIST *l1, LIST *l2, LIST *newl);
```

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

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



Πρόβλημα 2: mergeLists

- Η Συνάρτηση mergeLists ()

```
int mergeLists(LIST *l1, LIST *l2, LIST *newl) {  
    if ((l1 == NULL) || (l2 == NULL) || (newl == NULL)) return EXIT_FAILURE;  
    newl->head = mergeLoop(l1->head, l2->head);  
    newl->size = l1->size + l2->size;  
    return EXIT_SUCCESS;  
}
```

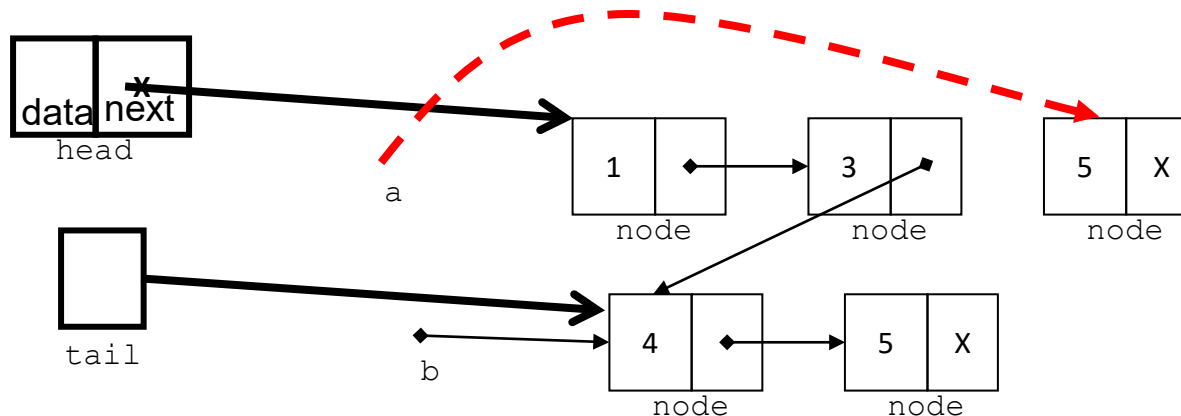
- Η Κλήση mergeLists () από το main ()

```
int main() {  
    LIST *l1, *l2, *l3;  
    l1 = l2 = l3 = NULL;  
    if (initList(&l1)==EXIT_FAILURE || initList(&l2)==EXIT_FAILURE || initList(&l3)  
    == EXIT_FAILURE) {  
        printf("Error: Unable to initialize list");  
        return EXIT_FAILURE;  
    } // fill l1, l2 with some values at this point ...  
    if (mergeLists(l1,l2,l3)==EXIT_FAILURE) { printf("..."); }  
}
```



Πρόβλημα 2: mergeLists

1. Έχουμε δυο μέτωπα λιστών a και b.
2. Θα δημιουργήσουμε **βοηθητικό κόμβο** head ο οποίος θα δείχνει στην κεφαλή της συγχωνευμένης λίστας.
3. Με **δείκτη** θα θυμόμαστε και το **τέλος** της συγχωνευμένης λίστας
 - έτσι ώστε να προσθέτουμε τον επόμενο στο tail->next:
4. Απεικόνιση Κατάστασης μετά την συγχώνευση 1, 3, 4:



Το **a** και **b** θα υποδεικνύουν τον επόμενο κόμβο κάθε λίστας

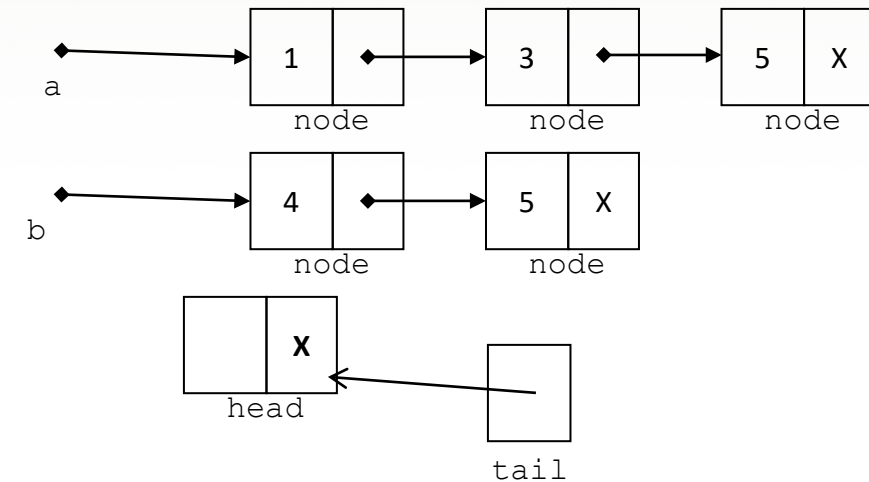
(Δεν χάσαμε αναφορά στο 3 εφόσον έδειχνε το tail πάνω του)
Το tail επιτελεί το ρόλο του prev δείκτη ενώ a, b είναι τα μέτωπα.



Πρόβλημα 2: mergeLists

Επαναληπτική Λύση

```
static NODE *MergeLoop(NODE *a, NODE *b) {  
    //Βοηθητικοί για αρχή, τέλος  
    NODE head;  
    NODE *tail = &head;  
  
    head.next = NULL;  
    // or tail->next = NULL  
    while (true) {  
        if (a == NULL) { // Φτάσαμε στο τέλος της a  
            tail->next = b; // ένωση υπόλοιπη λίστα b  
            break;  
        }  
        else if (b == NULL) { // Φτάσαμε στο τέλος της b  
            tail->next = a; // ένωση υπόλοιπη λίστα a  
            break;  
        }  
    }  
}
```

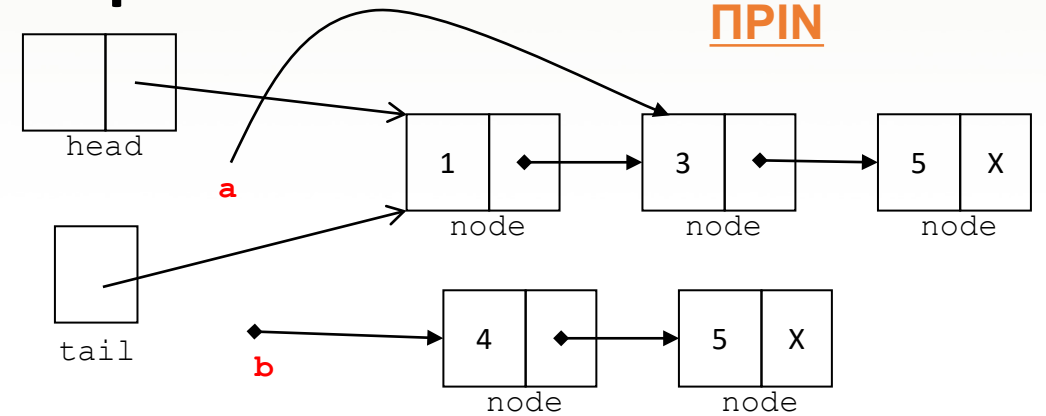


Πρόβλημα 2: mergeLists

Επαναληπτική Λύση

```
// Συνέχεια
//while (true) { ...
    if (a->data <= b->data) {
        tail->next = a;
        a = a->next;
    }
    // head.next now points
    // to the beginning of the
    // merged list (a,b are at the end)
```

1



Πρόβλημα 2: mergeLists

Επαναληπτική Λύση

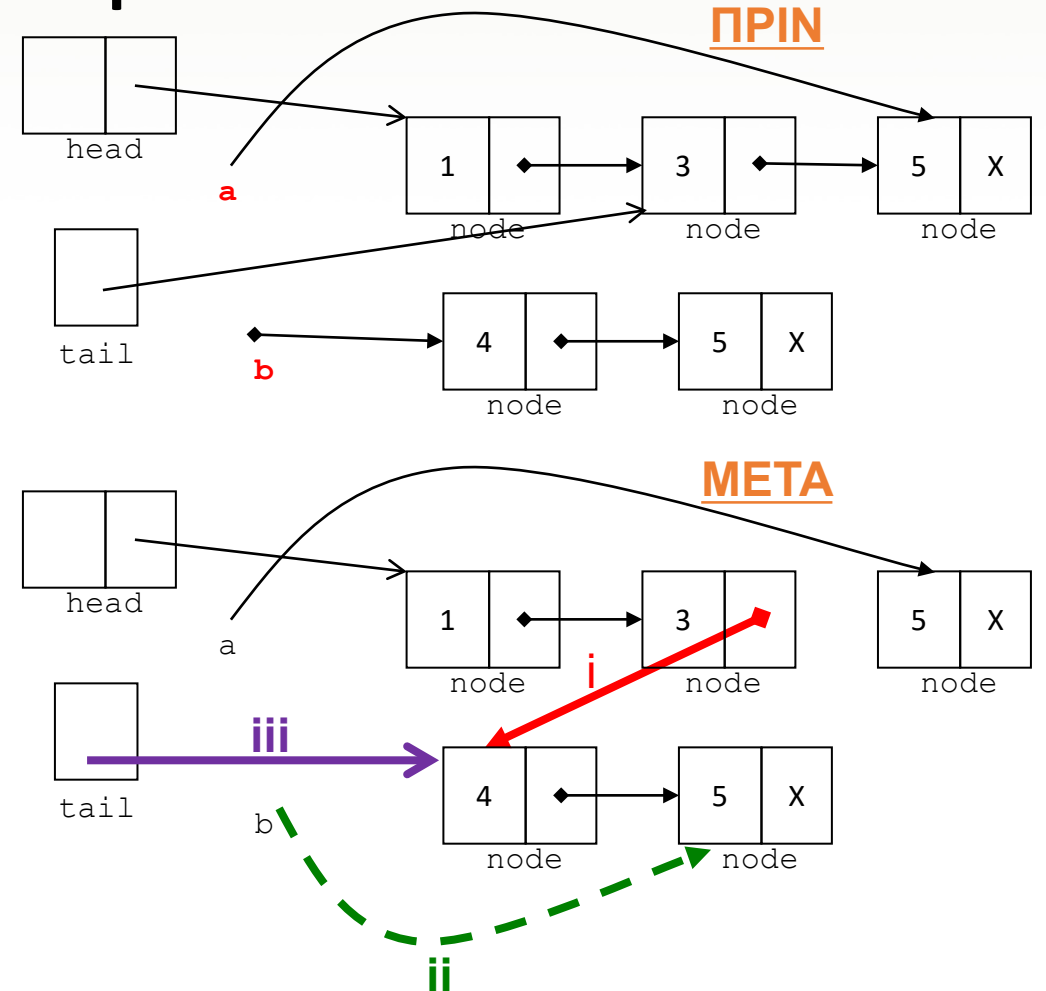
```

// συνέχεια
//while (true) { ...
  if (a->data <= b->data) {
    tail->next = a;
    a = a->next;
  } else {
    tail->next = b; // i
    b = b->next;   // ii
  }

  tail = tail->next; // iii
}
// head.next now points
// to the beginning of the
// merged list (a,b are at the end)
return head.next;
}

```

2



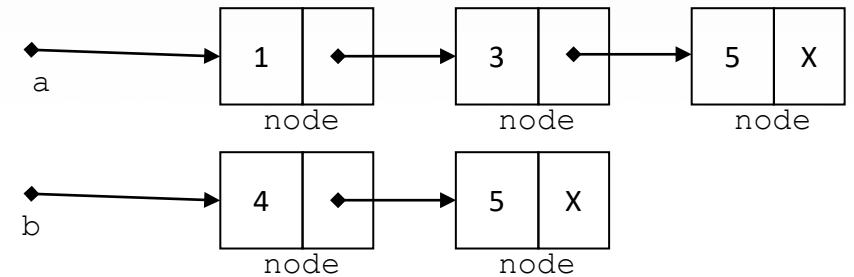
Πρόβλημα 2: mergeLists

Αναδρομική Λύση

```
static NODE *mergeRecursive(NODE *a, NODE *b) {
    // Base cases: either terminated.
    if (a == NULL) return b;
    else if (b == NULL) return a;

    // Pick either a or b, and recur
    if (a->data <= b->data) {
        a->next = mergeRecursive(a->next, b);
        return a;
    }
    else {
        b->next = mergeRecursive(a, b->next);
        return b;
    }
}

int mergeLists(const LIST *l1, const LIST *l2, LIST *newl) {
    if ((l1 == NULL) || (l2 == NULL) || (l3 == NULL)) return EXIT_FAILURE;
    newl->head = mergeRecursive(l1->head, l2->head);
    newl->size = l1->size + l2->size;
    return EXIT_SUCCESS;
}
```



Περιγραφή Προβλήματος **wordcount**

- **Ζητούμενο:** Πρόγραμμα που παίρνει σαν όρισμα το **όνομα ενός αρχείου** και μετρά τον αριθμό των εμφανίσεων όλων των **διαφορετικών λέξεων** που περιέχονται στο **αρχείο**.

- **Λύση: Δομή / Ψευδοκώδικας:**

Ανοιξε το αρχείο;

Δημιούργησε μια κενή λίστα;

while (δεν έχεις φτάσει στο τέλος του αρχείου)

if (η επόμενη λέξη υπάρχει ήδη στη λίστα)

 Αύξησε τον μετρητή που αντιστοιχεί στη λέξη κατά ένα;

else

 Πρόσθεσε κόμβο με τη λέξη στη λίστα;

Τύπωσε τα στοιχεία της λίστας;

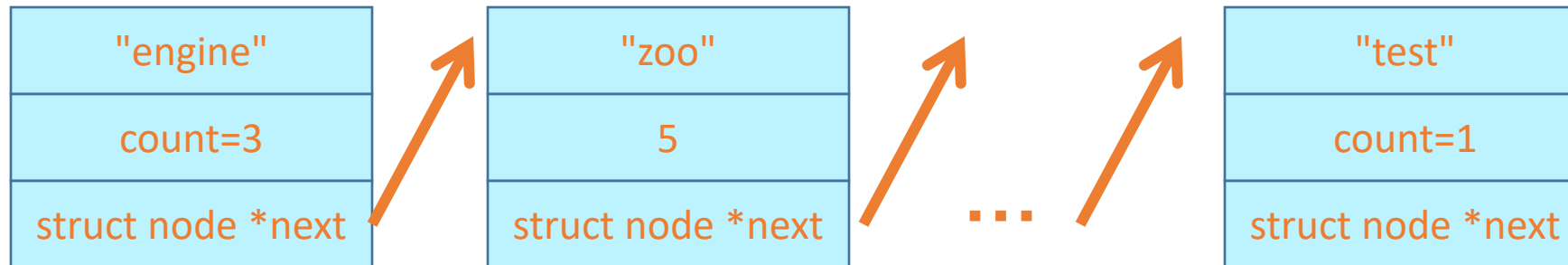
Κλείσε το αρχείο;



Διάγραμμα Λύσης `wordcount`

- **Επισήμανση:** Το σύνολο των πιθανών λέξεων του αρχείου δεν είναι εκ των προτέρων γνωστό.
 - Επομένως δεν μπορούμε να τις φυλάξουμε τα δεδομένα σε πίνακα τον οποίο εν συνεχεία θα επεξεργαστούμε (για να βρούμε την εμφάνιση των μοναδικών λέξεων).

- Ποια θα ήταν μια κατάλληλη δυναμική δομή;



- Μπορούμε όμως να διατηρούμε σε μια συνδεδεμένη λίστα τις διαφορετικές λέξεις του αρχείου **που έχουμε συναντήσει μέχρι στιγμής** (αρχικά μη-ταξινομημένη)
 - μαζί με ένα μετρητή για τον (μέχρι στιγμής) αριθμό των εμφανίσεων της λέξης.



Δομές Δεδομένων Λύσης **wordcount**

- Οι **κόμβοι** της λίστας θα περιέχουν τρία πεδία:

```
typedef struct node {  
    char word[MAXWORD];  
    int count;  
    struct node *next;  
} NODE;
```

- Ο **κόμβος** που ορίζει τη λίστα θα έχει τη μορφή:

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

Συναρτήσεις Λύσης `wordcount`

- Θα χρειαστούμε τις εξής χρήσιμες συναρτήσεις:
 - **`void *addlist(LIST *l, char *w) :`**
 - Η συνάρτηση αυτή παίρνει σαν παραμέτρους δείκτη προς τη λίστα "l" και δείκτη προς μια λέξη "w" και ψάχνει να βρεί τη λέξη "w" στη λίστα.
 - Αν υπάρχει, τότε αυξάνει το πεδίο count του κόμβου "w" κατά ένα, διαφορετικά δημιουργεί νέο κόμβο που αντιστοιχεί στη νέα λέξη "w".
 - **`void printlist(LIST *) :`**
 - Η συνάρτηση **τυπώνει τις λέξεις** της λίστας που δείχνεται από τον δείκτη που της δίνεται ως παράμετρος **μαζί με τον αριθμό των εμφανίσεων** κάθε **μιας από αυτές**.



Συναρτήσεις Λύσης `wordcount: H main ()`

```
#include <stdio.h>
#include <string.h>

#define MAXWORD 100

void addlist(LIST *, char *);
void printlist(LIST *);

int main(int argc, char *argv){
    LIST *list;
    char word[MAXWORD];
    FILE *inp;

    if (argc != 2) {
        printf("Error in the use of the program\n");
        exit(1);
    }
}
```



Συναρτήσεις Λύσης `wordcount: H main ()`

```
list.head = NULL;

inp = fopen(argv[argc-1], "r");

while(fscanf(inp, "%s", word) != EOF)
    if (isalpha(word[0])
        addlist(&list, word);

printlist(&list);

fclose(inp);
return 0;
}
```

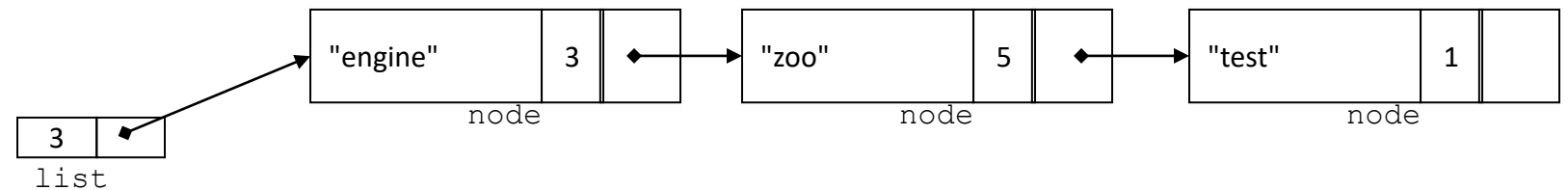
A value different from zero (i.e., true) if indeed is an alphabetic letter. Zero (i.e., false) otherwise.



Συναρτήσεις Λύσης `printlist()`

```
void printlist(LIST *l){
    NODE *p = l->head;

    while (p != NULL){
        printf("Word %s appears"
              "%d times\n",
              p->word, p->count);
        p = p->next;
    }
}
```



Συναρτήσεις Λύσης `addlist()` με αναδρομή

```
void addlist(LIST *l, char *w){
```

```
    l->top = addnode(l->head, w);
```

```
}
```

```
NODE *addnode(NODE *p, char *w){
```

```
    if (p == NULL){ // Δεν βρέθηκε τελικά => εισαγωγή
```

```
        p = (NODE *) malloc (sizeof(NODE));
```

```
        strcpy(p -> word, w);
```

```
        p->count = 1;
```

```
        p->next = NULL;
```

```
    }
```

```
    else if (strcmp(p -> word, w) == 0) // Στοιχείο Βρέθηκε
```

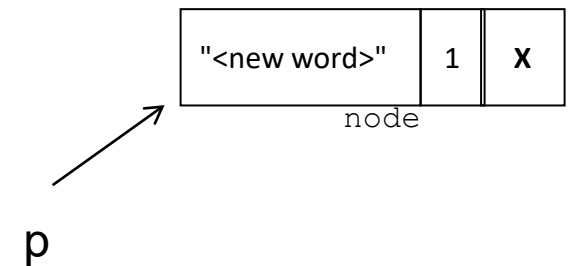
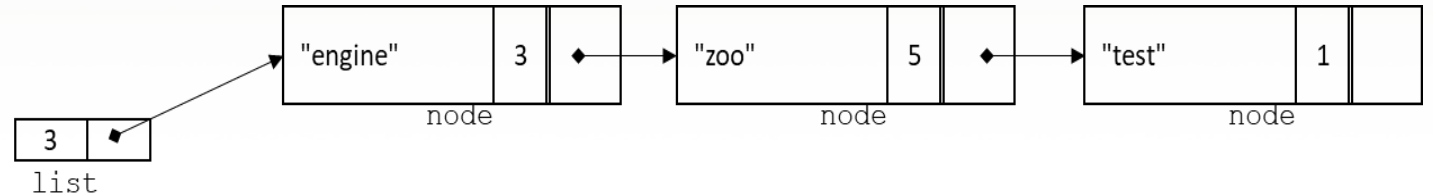
```
        p -> count ++;
```

```
    else // Στοιχείο ΔΕΝ Βρέθηκε ακόμη - αναδρομικά προχωρούμε
```

```
        p -> next = addnode(p -> next, w);
```

```
    return p;
```

```
}
```



Βελτιωμένες Δομές Δεδομένων Λύσης

- Παρατηρούμε πως ένα σημείο πιθανής **σπατάλης χώρου** είναι ότι σε κάθε κόμβο **δεσμεύουμε μνήμη** για λέξεις **100 χαρακτήρων**.
- Μπορούμε να αποφύγουμε αυτή τη **σπατάλη δεσμεύοντας** κάθε φορά **ακριβώς** όση **μνήμη** χρειαζόμαστε κάνοντας τις εξής αλλαγές στον κώδικά μας:
- Ορισμός κόμβου λίστας:

ΠΡΙΝ

```
typedef struct node{
    char word[MAXWORD];
    int count;
    struct node *next;
} NODE;
```

ΒΕΛΤΙΩΜΕΝΟ

```
typedef struct node{
    char *word;
    int count;
    struct node *next;
} NODE;
```



Βελτιωμένες Δομές & Συναρτήσεις `addnode ()`

```
NODE *addnode(NODE *p, char *w){  
  
    if (p == NULL){  
        p = (NODE *) malloc (sizeof(NODE));  
        p -> word = walloc(w);  
        p -> count = 1;  
        p -> next = NULL;  
    }  
    else if (strcmp(p -> word, w ) == 0)  
        p -> count ++;  
    else  
        p -> next = addnode(p -> next, w);  
    return p;  
}
```

Συνάρτηση που
παρουσιάζεται στην
επόμενη διαφάνεια



Βελτιωμένες Δομές & Συναρτήσεις `walloc()`

```
char *walloc(char *s){  
  
    char *p;  
  
    p = (char *) malloc (strlen(s)+1);  
    if (p != NULL)  
        strcpy(p,s);  
    return p;  
}
```



Βελτιωμένες Δομές & Συναρτήσεις `addNode ()`

- Κατά το ψάξιμο μιας λέξης στη λίστα, της λύσης 1, εκτελούμε **γραμμική διερεύνηση**
 - (δηλ. από αριστερά προς τα δεξιά μέχρι είτε να βρούμε τη ζητούμενη λέξη, είτε να φτάσουμε στο τέλος της λίστας).
- Αυτό δεν είναι και **τόσο αποδοτικό!**
 - Θα μπορούσαμε να βελτιώσουμε την απόδοση του προγράμματός μας διατηρώντας τη λίστα ταξινομημένη.
 - Έτσι αν κατά τη διερεύνηση κάποιας λέξης βρούμε λέξη στη λίστα που είναι λεξικογραφικά μεγαλύτερη από αυτή που ψάχνουμε μπορούμε να συμπεράνουμε ότι η λέξη δεν βρίσκεται στη λίστα.
 - επομένως δεν χρειάζεται να **ελέγξουμε τα υπόλοιπα στοιχεία** και η εισαγωγή κόμβου γίνεται άμεσα στο σημείο που βρισκόμαστε.

Βελτιωμένες Δομές & Συναρτήσεις `addNode ()`

```
NODE *addnode(NODE *p, char *w){
    NODE *q = NULL;
    int cond;

    if (p == NULL){ // Τέλος λίστας και δεν βρέθηκε
        q = (NODE *)malloc(sizeof(NODE));
        q -> word = walloc(w);
        q -> count = 1;
        q -> next = NULL;
    }
    cond = strcmp(p -> word, w);
    if (cond == 0) // λέξη "w" βρέθηκε
        (p->count)++;
    else if (cond < 0){ // λέξη p->word μικρότερη από "w"
        q = p;
        q->next = addnode(q->next, w);
    }
    else { // λέξη p->word μεγαλύτερη από "w"
        q = (NODE *) malloc (sizeof(NODE));
        q -> word = walloc(w);
        q -> count = 1;
        q -> next = p;
    }
    return q;
}
```

