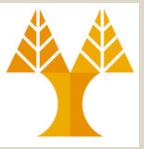
Τμήμα Πληροφορικής
Πανεπιστήμιο Κύπρου



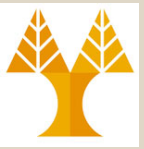
ΕΠΛ232 – Προγραμματιστικές Τεχνικές και Εργαλεία

Διάλεξη 13: Δομές Δεδομένων II (Ταξινομημένες Λίστες)

Δημήτρης Ζεϊναλιπούρ

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

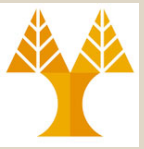
Περιεχόμενο Διάλεξης 13



- **Στοίβα, Ουρά και Ταξινόμηση**
  - Επανάληψη Στοίβας και Ουράς
  - Αδυναμία Υποστήριξης Ταξινόμησης
- **Ταξινομημένες Λίστες**
  - Δηλώσεις και Αρχικοποίηση (στατικά, δυναμικά, σε συνάρτηση, δείκτη-δείκτη)
  - Διαδικασίες printList(), printListRecursive()
  - Διαδικασίες insert(), insertRecursive()
  - Διαδικασίες delete(), deleteRecursive()

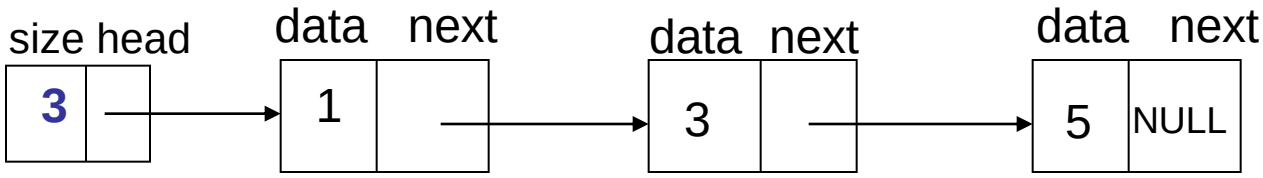
Επόμενη Διάλεξη

Ταξινομημένες Λίστες

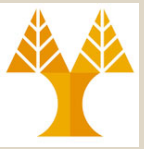


- ***Ταξινομημένα τα Δεδομένα σε κάποια σειρά*** (π.χ., αύξουσα στα ακόλουθα παραδείγματα) και από την οποία να μπορούμε εύκολα να προσθέτουμε και να αφαιρούμε στοιχεία.

- Έτσι, για παράδειγμα αν εφαρμόζαμε διαδοχικά τις εντολές: **εισαγωγή του 5**, **εισαγωγή του 1 και εισαγωγή του 3** θα θέλαμε η λίστα μας να είχε τη μορφή:



Ταξινομημένες Λίστες (Δηλώσεις και Αρχικοποίηση)



- Για την υλοποίηση μιας δομής τύπου **Ταξινομημένη Λίστα** (όπως και στις στοίβες και ουρές) απαιτείται η παρακάτω δήλωση κόμβων:


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

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

- Οι ορισμοί λοιπόν και οι κλήσεις (4 εναλλακτικοί τρόποι):

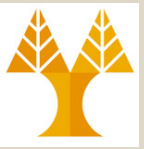
A) Στατικά στο main()

```
LIST list;
list.head = NULL;
list.size = 0;
foo(&list);

void foo(LIST *list);
```

B) Δυναμικά στο main()

```
LIST *list = NULL;
list = (LIST *)
          malloc(sizeof(LIST));
list->head = NULL;
list->size = 0;
foo(list);
```

Ταξινομημένες Λίστες (Δηλώσεις και Αρχικοποίηση)



Γ) Δυναμικά σε Συνάρτηση (με return pointer):

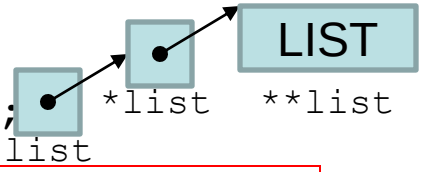
Κατά την έξοδο, η αναφορά list χάνεται όχι όμως ο χώρος που δεσμεύτηκε με την malloc (η διεύθυνση του οποίου επιστρέφεται)

```
LIST *initList() {
   LIST *list = (LIST *) malloc(sizeof(LIST));
    if (list == NULL) return NULL;
   list->head = NULL;
   list->size = 0;
   return list;
}
```

```
int main() {
   LIST *list = initList();
   ...
}
```

Δ) Δυναμικά σε Συνάρτηση (με δείκτη-σε-δείκτη):

```
int initList2(LIST **list) {
       *list = (LIST *) malloc(sizeof(LIST));
       if ((*list) == NULL) return EXIT_FAILURE;
       (*list)->head = NULL;
       (*list)->size = 0;
       return EXIT_SUCCESS;
}
```

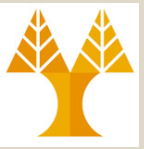



```
int main() {
   LIST *list = NULL;
   initList2(&list);
   ...
}
```

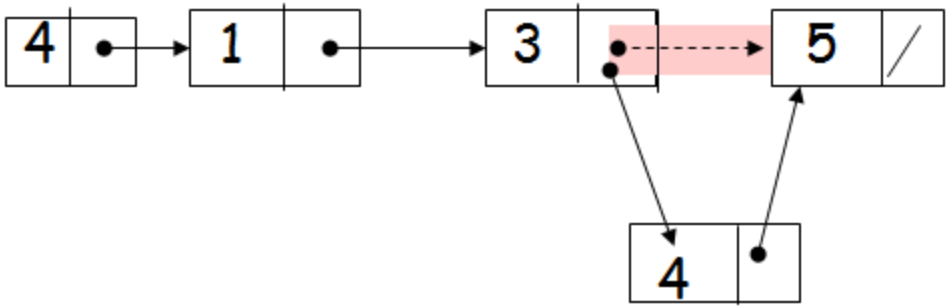
Ταξινομημένες Λίστες (Λειτουργίες / Διαδικασίες)

- Τώρα απομένει να ορίσουμε τρείς διαδικασίες:
  - **`printlist`**: τυπώνει όλα τα στοιχεία της λίστας l.
    ```
    void printlist(LIST *l);
    ```
  - **`insert`**: εισαγάγει ένα στοιχείο x μέσα στη λίστα l.
    ```
    void insert(LIST *l, int x);
    ```
  - **`delete`**: αφαιρεί κάποιο στοιχείο x (αν υπάρχει) από τη λίστα l.
    ```
    void delete(LIST *l, int x);
    ```
- Οι πιο πάνω λειτουργίες μαζί με τις συνοδευτικές λειτουργίες δημιουργίας, καταστροφής της λίστας πρέπει να τοποθετηθούν σε αρχεία `list.c` και `list.h`
  - Μαζί με τον σχετικό **οδηγό δοκιμής** **`#ifdef DEBUG … #endif`**.
  - Το `main` θα περιέχει ένα switch που μας επιτρέπει να καλέσουμε τις διαδικασίες ταξινομημένης λίστας.
  - Δεν θα μας απασχολήσει στο παρόν στάδιο το return code των συναρτήσεων αυτών.

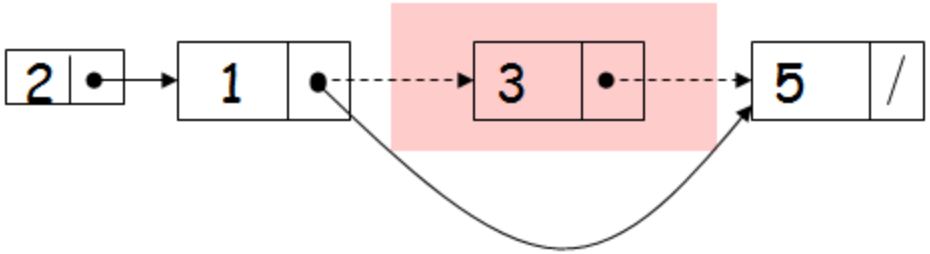
Ταξινομημένες Λίστες (Διαγραμματική Απεικόνιση)



- **Εισαγωγή** του **4** στη **λίστα:**



- **Εξαγωγή του 3** από τη λίστα:



Περιγράψετε το διάγραμμα μιας λύσης στο πιο πάνω πρόβλημα. Τι οριακές περιπτώσεις εντοπίζετε;

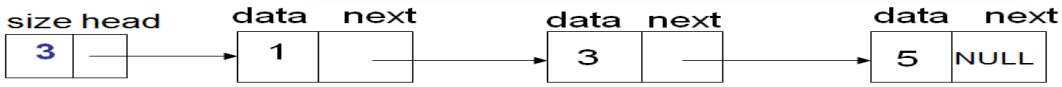
Ταξινομημένες Λίστες (Συνάρτηση Οδηγού Χρήσης)

```
#ifdef DEBUG
int main(){
  int x = 0;  /* User Function Choice*/
  int y;      /* Value to be inserted/deleted */
  LIST *list = initList();

  while (x != 4){

    printf("You have the following options:\n");
    printf("\t 1\t: inserting an item in the list\n");
    printf("\t 2\t: deleting an item from the list\n");
    printf("\t 3\t: printing the list elements\n");
    printf("\t 4\t: exiting from the system\n");
    scanf("%d", &x);

    // συνέχεια επόμενη διαφάνεια
```

Ταξινομημένες Λίστες (Συνάρτηση Οδηγού Χρήσης)

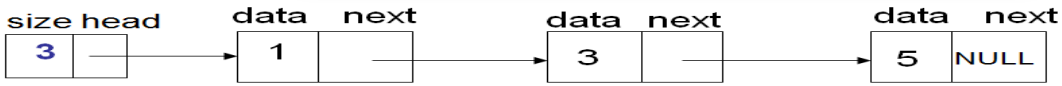
```
 switch (x){
   case 1: printf("Enter number to be inserted: ");
           scanf("%d", &y);
           insert(list, y);
           break;
   case 2: printf("Give me the number to be deleted: ");
           scanf("%d", &y);
           delete(list, y);
           break;
   case 3: printlist(list);
           break;
   case 4: printf("Goodbye!\n");
           break;
  default: printf("Wrong entry - try again!\n");
           break;
  }
 }
 return 0;
}
#endif
```

Ταξινομημένες Λίστες (Συνάρτηση `printlist`)




```
void printlist(LIST *l) {
  NODE *p = NULL; // copy pointer
  if ((l == NULL) || (l->size == 0)) {
        printf("The list is empty\n");
        return;
  }
  p = l->head;
  while (p != NULL){
      printf("%d ", p->data);
      p = p->next;
  }
  printf("\n");
}
```

Ταξινομημένες Λίστες (Αναδρομική Συνάρτηση `printlist`)



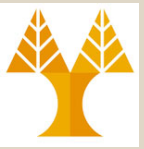

```
void printlist(LIST *l){
  if ((l == NULL) || (l->size == 0))
      printf("The list is empty\n");
  else
      printnode(l->head);
}
```

"Εσωτερική συνάρτηση" με εμβέλεια αρχείου

```
static void printnode(NODE *p){
  if (p != NULL){
      printf("%d", p->data);
      printnode(p->next);
   }
}
```

Αναδρομή Χωρίς return στην οπισθοχώρηση

Πρόβλημα 1: `count`



- Γράψετε συνάρτηση στη γλώσσα C η οποία παίρνει ως όρισμα ένα δείκτη σε λίστα και η οποία επιστρέφει το μέγεθος της λίστας.
- **Πρότυπο Συνάρτησης:**

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

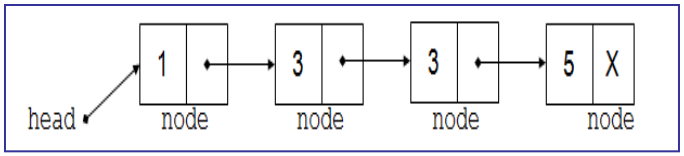
```
typedef struct node {
   int data;
   struct node *next;
} NODE;
```

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

- **Κλήση Συνάρτησης**

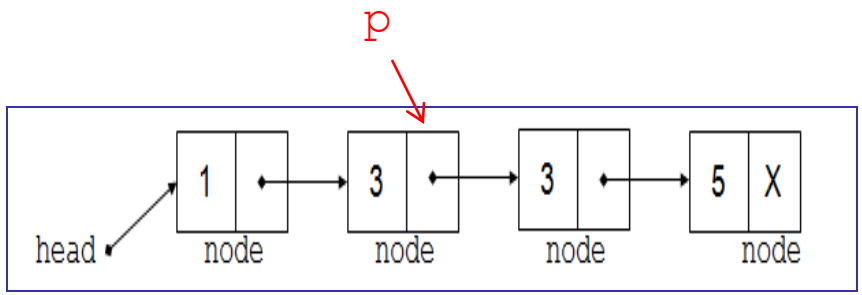
```
LIST *list;
...
printf("%d", count(list));
```

➔ τυπώνει 4



Πρόβλημα 1: count (Επαναληπτική Λύση)

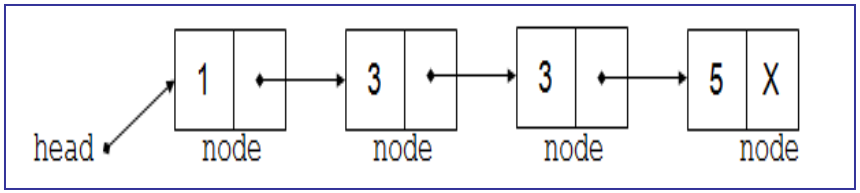
```
static int countList(NODE *p) {
    int count = 0;
    while (p != NULL) {
        count++;
        p = p->next;
    }
    return count;
}
```




```
int count(LIST *l) {
   if (l == NULL) return 0;
   return countList(l->head);
}
```

Πρόβλημα 1: count (Αναδρομική Λύση 1)

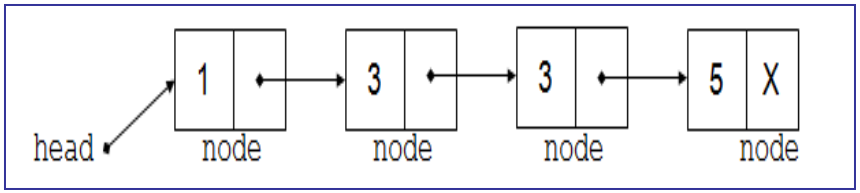
```
static void countListRecursive(NODE *p, int *count) {
     if (p == NULL) {
          return 0;
     }
      (*count)++;
     countListRecursive(p->next, count);
}

int count(LIST *l) {
    if (l == NULL) return 0;
    int count = 0;
    countListRecursive(l->head, &count);
    return count;
}
```



Αναδρομή ΧΩΡΙΣ return στην οπισθοχώρηση και τιμή δια αναφοράς

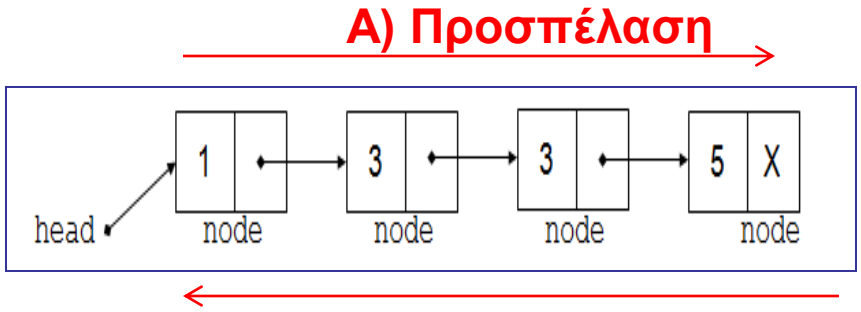
Πρόβλημα 1: count (Αναδρομική Λύση 2)

```
static int countListRecursive(NODE *p) {
    if (p == NULL) {
        return 0;
    }
    return 1 + countListRecursive(p->next);
}

int count(LIST *l) {
   if (l == NULL) return 0;
   return countListRecursive(l->head);
}
```



Αναδρομή ΜΕ return στην οπισθοχώρηση

Σύνοψη Αναδρομικών Προσεγγίσεων



Προσπέλαση (void στη Οπισθοχώρηση)

- Εκτύπωση Λίστας: **`void printnode(NODE *p);`**
- Count με Εκτύπωση: **`void countLR(NODE *p, int count);`**
- Count με Δια'αναφοράς: `void` **`countLR(`**`NODE *p, int *count);`
- Όμοια Υλοποιούνται οι : `Sum, Max, Min, Avg (δηλ., Sum/Count)`
- Πολλαπλές με Δια'αναφοράς:

  `void` **`countLR(`**`NODE *p, int *sum, int *count);`

Προσπέλαση + Οπισθοχώρηση

- Count με Επιστροφή: **`int countLR(NODE *p);`**