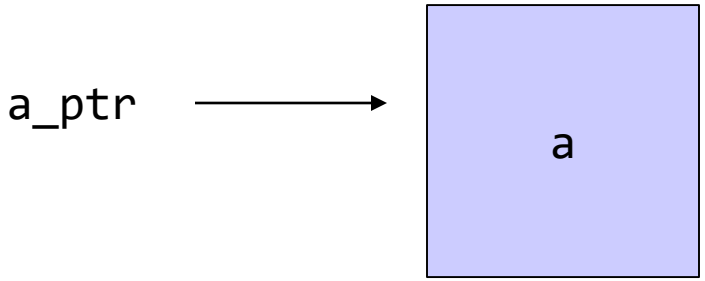
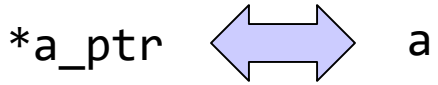
# *Διαδικαστικός Προγραμματισμός*

Βασίλης Παλιουράς
paliuras@ece.upatras.gr

# Βασικό για τα παρακάτω…

```
T * a_ptr;

a_ptr = &a;
```

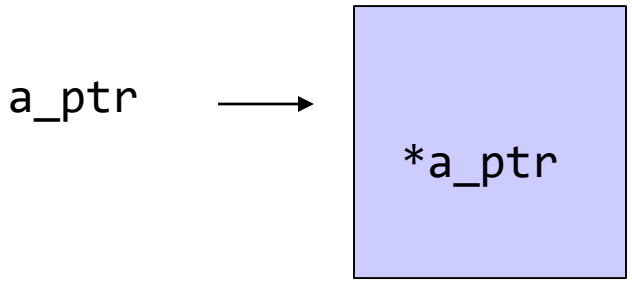


# Θεμελιώδης τεχνική

Μνήμη που δεσμεύεται για παράδειγμα με `malloc` ή άλλο κατάλληλο τρόπο

```
T * a_ptr;

a_ptr = malloc (sizeof (T)) ;
```




*a_ptr ως μεταβλητή τύπου T

*a_ptr ερμηνεύει τα δεδομένα στην περιοχή ως τύπου T

*a_ptr για να διαβάσουμε/γράψουμε – επεξεργαστούμε αυτά τα δεδομένα

```
{T x;
 …κώδικας…

F(x);
 …κώδικας…

}

Y F(T x) {
  Y y;
  …κώδικας…
  x = έκφραση;
  return y;
  }
```

```
{T x;
 …κώδικας…

F(&x);
 …κώδικας…

}

Y F(T * x) {
  Y y;
  …κώδικας…
  (*x) = έκφραση;
  return y;
  }
```

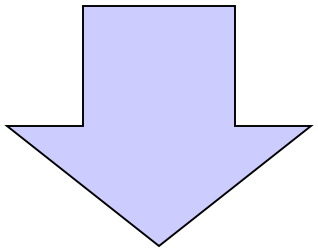
Τ οποιοσδήποτε τύπος

Παράδειγμα

```
typedef char **  T;
```

# Απλοποιημένος συμβολισμός

- `struct mystruct *test_ptr;`
- `(*test_ptr).next`



- `test_ptr->next`

# Τύποι λίστας στο mytypes.h

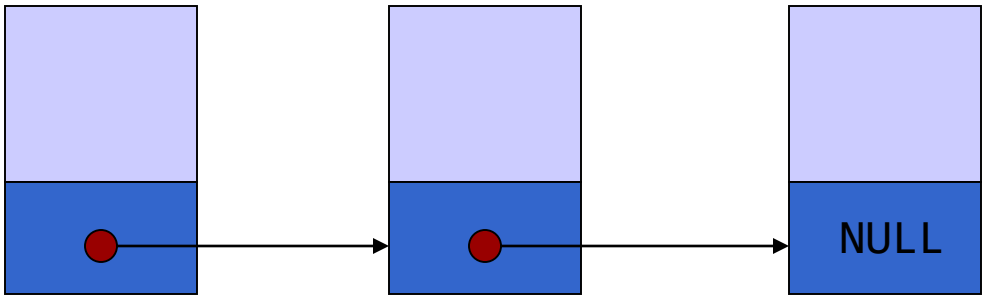
- Listelement
  - Στοιχείο λίστας (**struct** listelement)
  - **typedef struct** listelement Listelement ;
- Listelement_ptr
  - Δείκτης σε στοιχείο λίστας. Ίδιο με
    - **struct** listelement *
    - Listelement *
    - **typedef** Listelement * Listelement_ptr;
- List
  - Δείκτης σε στοιχείο λίστας
  - **typedef** Listelement * List;
  - Μεταβλητές αυτού του τύπου δείχνουν
    - Στο πρώτο στοιχείο της λίστας ή
    - Έχουν την τιμή NULL (κενή λίστα).

# Δημιουργία στοιχείου

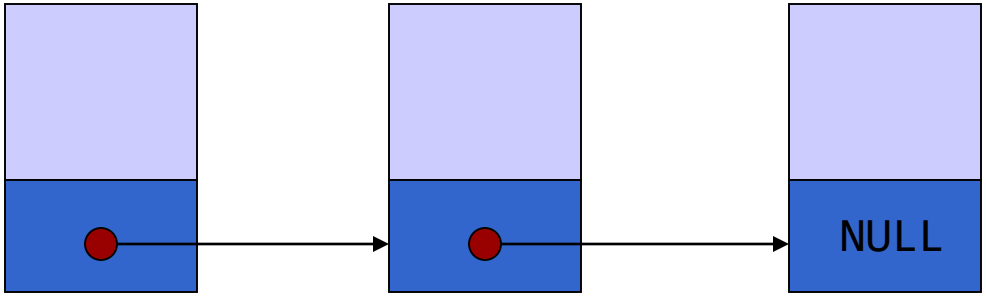
```
#include <string.h>
#include <stdlib.h>
#include "mytypes.h"

Listelement_ptr createnewelement(char word[], int number) {
        Listelement_ptr newelement_ptr;

        newelement_ptr =  malloc ( sizeof (Listelement));
        strcpy(newelement_ptr->name, word);
        newelement_ptr -> age = number;
        newelement_ptr -> next = NULL;

        return newelement_ptr;

}
```

# Διατρέχει όλα τα στοιχεία της λίστας

```
#include <stdio.h>
#include "mytypes.h"

void reportlist(List const alist) {
        Listelement_ptr iterator= alist;
        for (; iterator != NULL ; iterator = iterator->next)
         {
                printf( "name: %s\n", iterator->name);
                printf("age: %d\n", iterator->age);
         }
}
```



# Προσθήκη στοιχείου στο τέλος λίστας

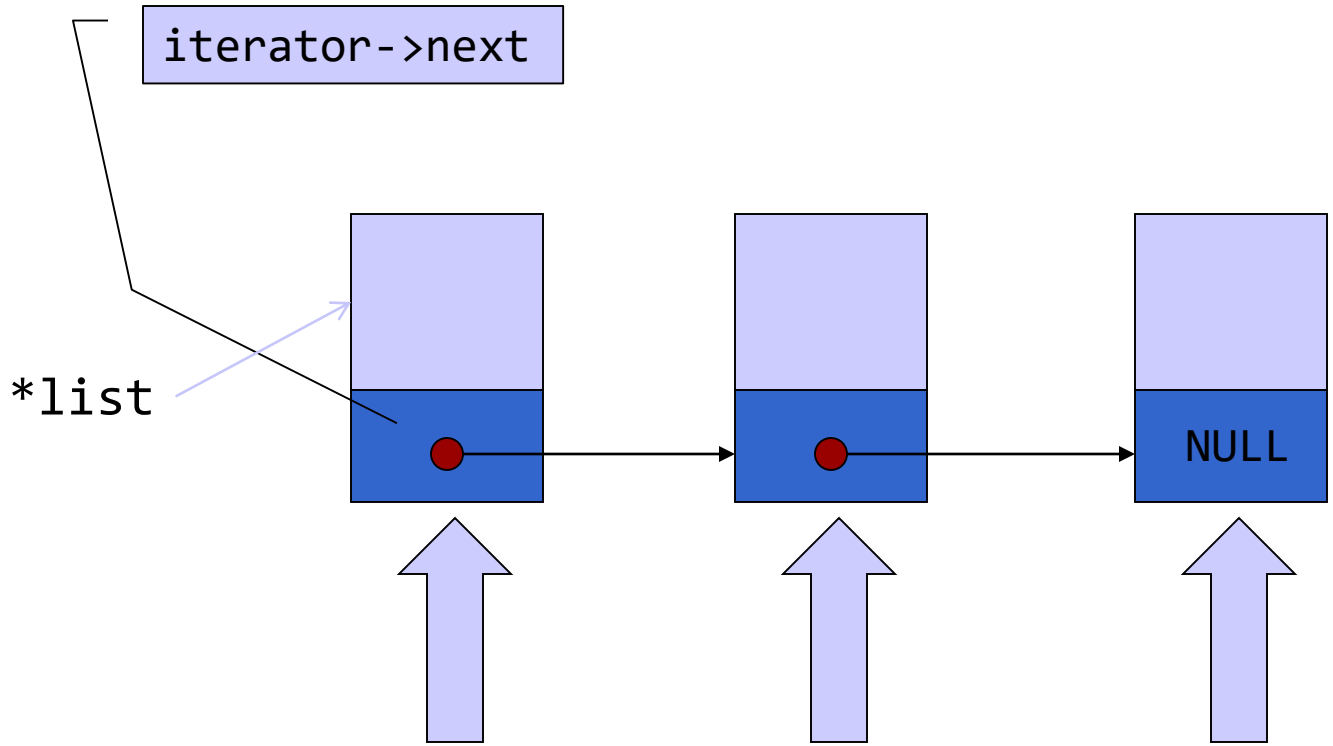



Παράμετρος δείκτης σε λίστα ή διπλός δείκτης σε στοιχείο

```
#include <stdio.h>
#include "mytypes.h"

void addtolist (List *list, Listelement_ptr elem_ptr ) {
        Listelement_ptr iterator = *list;
        if (*list==NULL) /* handle empty list*/
           *list = elem_ptr;
        else  { /* if not empty, move to last element list*/
                  for (; iterator->next != NULL; iterator = iterator->next);
                  /* append element to list */
                  iterator->next = elem_ptr;
        }

}
```

Όταν βγει από τον βρόχο, ο iterator δείχνει στο τελευταίο στοιχείο

```
Listelement_ptr iterator = *list;
```




Αφού το iterator->next γίνει NULL, ο βρόχος τερματίζεται.

Το iterator δείχνει στο Τελευταίο στοιχείο.

```
for (; iterator->next != NULL ; iterator = iterator->next)
```

# Διαχείριση με συναρτήσεις

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "mytypes.h"
Listelement_ptr createnewelement(char *, int);
Listelement_ptr findelementbyname(List, char []);
void reportlist( List const);
void addtolist(List *, Listelement_ptr);
int deleteelement(List *, Listelement_ptr);
int main(void) {
          /* Initialize */
          Listelement_ptr iterator=NULL, element_ptr, previous_ptr;
          List nameslist = NULL;
          char name[50];
          /* create elements and put them to list*/
          element_ptr = createnewelement("Ntina", 10);
          addtolist (&nameslist, element_ptr);
          element_ptr = createnewelement("Giannis", 5);
          addtolist( &nameslist, element_ptr);
          element_ptr = createnewelement("Maria", 7);
          addtolist( &nameslist, element_ptr);
          reportlist(nameslist);
          /* find, delete, report */
          do {
             printf("name: ");
             scanf("%s", name);
             element_ptr = findelementbyname(nameslist, name);
             if (element_ptr)  {
                       printf("found it. data: %d\n", element_ptr->age);
                       if (deleteelement(&nameslist, element_ptr) == 0)
                                          printf("...deleted.\n");
                       reportlist(nameslist);
                    }
          } while (nameslist!=NULL) ;
 return 0;
}
```

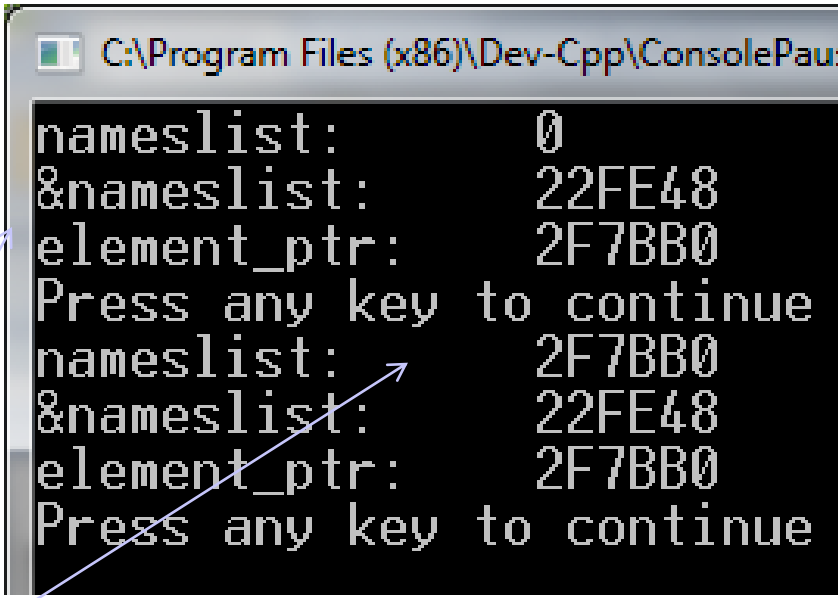
```
Listelement_ptr findelementbyname(List mylist, char name[]) {
        Listelement_ptr iterator;

        for (iterator = mylist; iterator != NULL; iterator = iterator -> next) {
                if (strcmp(iterator->name,name)==0 ) return iterator;
        }

        return NULL;
};
```

# Λίστα ως παράμετρος με αναφορά

```
void addtolist (List *, Listelement_ptr );
List nameslist = NULL;
Listelement_ptr = element_ptr;
element_ptr = createnewelement("Ntina", 10);
addtolist (&nameslist, element_ptr);
```

Τιμές **πριν** την εκτέλεση της addtolist

Τιμές **μετά** την εκτέλεση της addtolist




- Η `nameslist` αποθηκεύεται στη θέση 22FE48.
- Η τιμή της `nameslist` αλλάζει μετά την κλήση της `addtolist`. (όχι η θέση της!)
- Τύπος της `nameslist`: `List` Τύπος της θέσης της: `List *`

Θέση της namelist ( ή &nameslist)

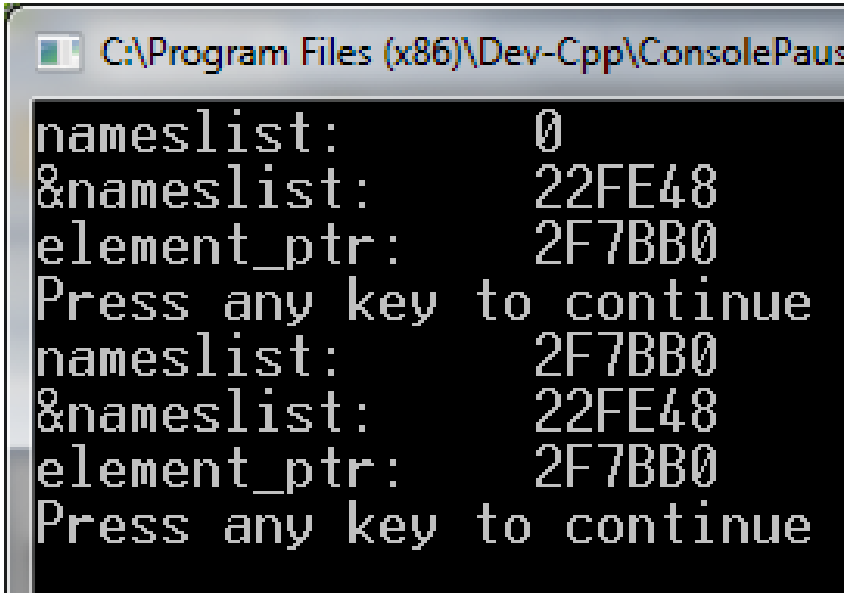
22FE48

Τιμή της namelist

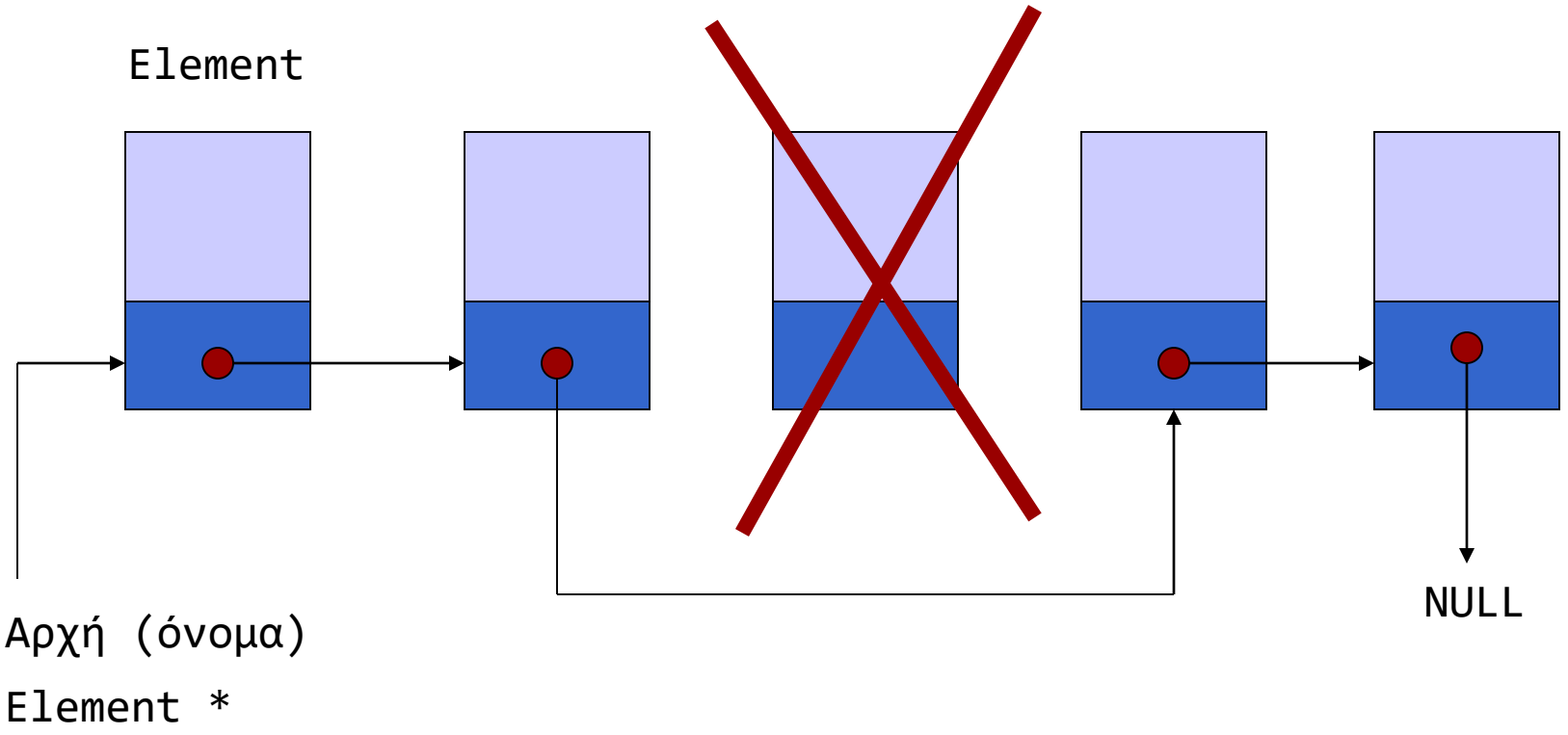
2F7BB0

Πριν την κλήση της addtolist

Μετά την κλήση της addtolist



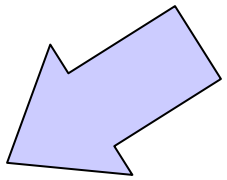
# Διαγραφή στοιχείου




Τι γίνεται με τη μνήμη την οποία το στοιχείο καταλάμβανε;

# Διαγραφή στοιχείου

Παράμετρος: δείκτης σε λίστα ή διπλός δείκτης σε στοιχείο λίστας



```
#include <stdio.h>
#include <stdlib.h>
#include "mytypes.h"

int deleteelement(List *alist, Listelement_ptr element) {
        Listelement_ptr  iterator = *alist, todelete_ptr;
        if (element == NULL || *alist == NULL)
           return -1 ;
        if (element== *alist) { /* if required, delete first element */
           *alist = element->next ;
           free(element);
        }
        else { /* find the element before the one to be deleted */
                  for(; iterator->next != element; iterator = iterator->next) ;
                  iterator->next = element->next;
                  free(element);
        }
        return 0;
}
```

# Διαγραφή με πρόβλεψη το στοιχείο να μην υπάρχει στη λίστα

```
#include <stdio.h>
#include <stdlib.h>
#include "mytypes.h"
int deleteelement(List *alist, Listelement_ptr  todelete_ptr) {
 Listelement_ptr  iterator = *alist;

if (todelete_ptr == NULL || *alist == NULL)
  return -1 ;
 if (todelete_ptr == *alist) {
   *alist = todelete_ptr->next ;
    free(todelete_ptr);
   }
 else {

    for(; iterator !A NULL && iterator->next != todelete_ptr ;
                                    iterator = iterator->next) ;
    if (iterator!=NULL) {
          iterator->next = todelete_ptr->next;
          free(todelete_ptr);}
    else {
          printf("element not in list");
          return -1;
          }
     }
  return 0;
}
```

Όσο δείχνεις σε στοιχείο της λίστας (iterator!=NULL) ΚΑΙ το επόμενο στοιχείο δεν είναι το προς διαγραφή (iterator->next != todelete_ptr), πήγαινε στο επόμενο στοιχείο

Αν μετά το βρόχο το iterator δείχνει σε στοιχείο (!=NULL), αυτό είναι το ακριβώς προηγούμενο από το προς διαγραφή.

# Διαγραφή με πρόβλεψη το στοιχείο να μην υπάρχει στη λίστα

```
#include <stdio.h>
#include <stdlib.h>
#include "mytypes.h"
int deleteelement(List *alist, Listelement_ptr  todelete_ptr) {
 Listelement_ptr  iterator = *alist;

if (todelete_ptr == NULL || *alist == NULL)
  return -1 ;
 if (todelete_ptr == *alist) { /* if required, delete first element*/
   *alist = todelete_ptr->next ;
    free(todelete_ptr);
   }
 else { /* find the element before the one to be deleted or
          continue until no more elements in list */
    for(; iterator != NULL && iterator->next != todelete_ptr ;
                                        iterator = iterator->next) ;
    if (iterator!=NULL) {
          iterator->next = todelete_ptr->next;
          free(todelete_ptr);}
    else {
          printf("element not in list");
          return -1;
          }
     }
  return 0;
}
```

Πρώτα χρησιμοποιώ το todelete_ptr
Μετά αποδεσμεύεται με free

Τι θα γίνει αν γράψουμε
`iterator->next!=todelete_ptr && iterator !=NULL`
αντί για
`iterator!=NULL && iterator->next!=todelete_ptr`

# Χρήσιμα σημεία σε συναρτήσεις επεξεργασίας λίστας

- Η λίστα ως παράμετρος
  - Αναφερόμαστε σε λίστα με τη διεύθυνση του πρώτου στοιχείου
  - Μερικές συναρτήσεις αλλάζουν τη διεύθυνση του πρώτου στοιχείου (προσθέτουν, διαγράφουν, …)
  - Άλλες όχι (εκτύπωση, καταμέτρηση, αναζήτηση…)
- Πρώτα χρησιμοποιώ, μετά διαγράφω
- Βρίσκω και καλύπτω ειδικές περιπτώσεις
  - Ενδεικτικά: άδεια λίστα, πρώτο στοιχείο, κενό στοιχείο ως είσοδος

# Αναδρομική αναζήτηση σε λίστα

```
#include <string.h>
#include "mytypes.h"

Listelement_ptr recfind(List alist, char name[]) {
if (alist == NULL)
   return NULL;
else
       if (!strcmp(alist->name, name))
          return alist;
       else
          return recfind(alist->next, name);
}
```

# Άλλο Παράδειγμα

- Παραγωγή ψευδοτυχαίων αριθμών και καταχώρηση σε απλά διασυνδεδεμένη λίστα της πληροφορίας
- Στη λίστα αυτή κάθε αριθμός καταχωρείται μόνο μία φορά.
  - ποιοι αριθμοί εμφανίστηκαν και
  - πόσες φορές ο καθένας.

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
        int num;
        int occur;
        struct node * next;
} ;
typedef struct node Node;
typedef Node * List;

Node *create(int num);
void report(List lst);
List update(List lst, int n);


 int main(void) {
        int s ;
        int i;
        List mylst = NULL;

        for (i = 0 ; i<10; i++) {
                s = ((double) rand() / RAND_MAX ) * 5;
                printf("before: %p ", (void *) mylst);
                mylst = update(mylst, s);
                printf("after: %p\n", (void *) mylst);
        }

        report(mylst);

        return EXIT_SUCCESS ;
 }
```

```
List update (List mylst, int n) {
          List temp_list = mylst;
          Node * iter;

          if (mylst == NULL) {
                    temp_list = create(n);
                    return temp_list;
          }
          for (iter = temp_list;
                              iter->next != NULL;
                                        iter = iter -> next) {
                                                  if (iter -> num == n) {
                                                            (iter -> occur)++;
                                                            return temp_list;
                                                            }
                                                  }
          if (iter->num==n) {   (iter->occur)++;
                              return temp_list;
                              }

          iter->next = create(n);

          return temp_list;
}
```

```
Node * create (int n) {
            Node * tmp;

            tmp = malloc( sizeof (Node));
            tmp -> num = n;
            tmp -> occur = 1;
            tmp -> next = NULL;
            return tmp;
 }
```

```
void report(List lst) {
           Node * iter;

           for (iter = lst ; iter != NULL; iter = iter -> next){
                      printf("%d (%d):", iter->num, iter->occur);
           }
           return;
}
```