ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά μαθήματα ΠΠ

ΑΡΧΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Κεφάλαιο 16

Επιμέλεια:
Βασίλης Παλιουράς , Αναπληρωτής Καθηγητής
Ευάγγελος Δερματάς , Αναπληρωτής Καθηγητής
Σταύρος Νούσιας , Βοηθός Ερευνητή
Πολυτεχνική Σχολή
Τμήμα Ηλεκτρολόγων Μηχανικών & Τεχνολογίας Υπολογιστών

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τυπου άδειας χρησης, η άδεια χρησης αναφερεται ρητώς.

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου των διδασκόντων καθηγητών.
- Το έργο «Ανοικτά Ακαδήμαϊκά Μαθήματα στο Πανεπιστήμιο Πατρών» έχει χρηματοδοτηθεί μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκη Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικους πόρους.

Ανάπτυξη

Το παρόν εκπαιδευτικό υλικό αναπτύχθηκε στο τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών του Πανεπιστημίου Πατρών

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Σκοπός

Στόχος της παρακάτω ενότητας είναι η εισαγωγη σε β ασικες αρχες και εννοιες των Αρχων Προγραμματισμ ού.

Σχετικά με το εργαστήριο 3

- Μια προσέγγιση
  - Διάβασε κείμενο από αρχείο σε πίνακα
  - Διάβασε λεξικό από αρχείο σε πίνακα
  - Τροποποίησε πίνακες
  - Γράψε πίνακες σε αρχείο
- Άλλη προσέγγιση
  - Προσθέτω λέξεις στο λεξικό κατευθείαν στο αρχεί ο

Δυναμική διαχείριση μνήμης στη C

- Δέσμευση μνήμης:
  - **void** *malloc(size_t size);
  - Επιστρέφει δείκτη σε εξασφαλισμένη περιοχή μεγέθους size bytes ή NULL αν δεν υπάρχει τέτοια.
- Απελευθέρωση μνήμης:
  - **void** free(**void** *pointer);

Πώς δουλεύει ο μηχανισμός;

- Χρησιμοποιεί
  - δεδομένα στο heap
  - Λεπτομερή διαχείριση ανά block
    - Διεύθυνση αρχής
    - Μέγεθος
- Μοιράζεται πληροφορία μεταξύ διαφορετικών συναρ τήσεων
  - malloc ( ) , free ( )
  - Πώς γίνεται αυτό;

Παράδειγμα

```
#include <stdio.h>
#include <stdlib.h>
#define N 10

main ( ) {

char matrix[N];

scanf("%s", matrix);

printf("Hello %s!\n", matrix);


}
```

Παράδειγμα

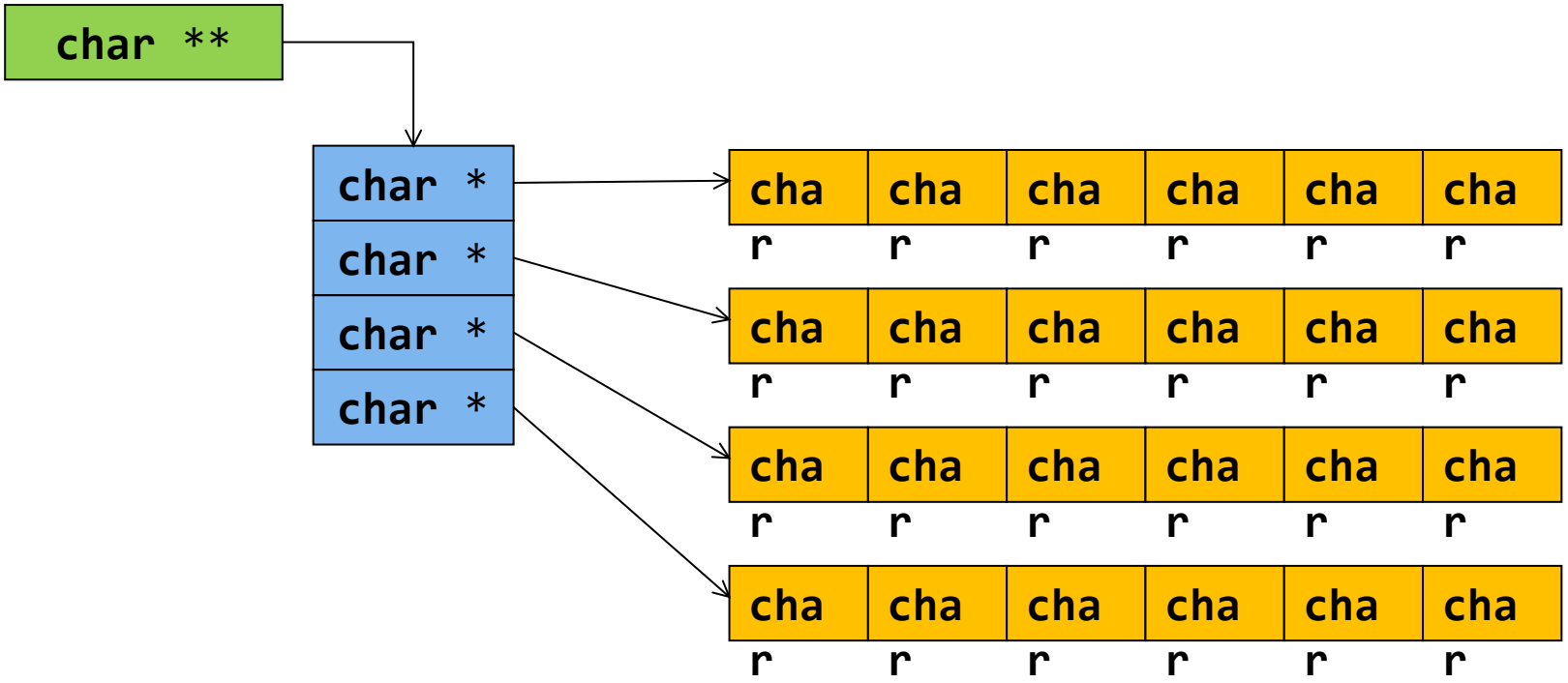
```
#include <stdio.h>
#include <stdlib.h>
#define N 10

main ( ) {

char matrix[N];
char *dynamicdata;

scanf("%s", matrix);

printf("Hello %s!\n", matrix);

dynamicdata = (char *) malloc( N * sizeof (char));

scanf("%s", dynamicdata);

printf("Hello dynamic %s!", dynamicdata);


}
```

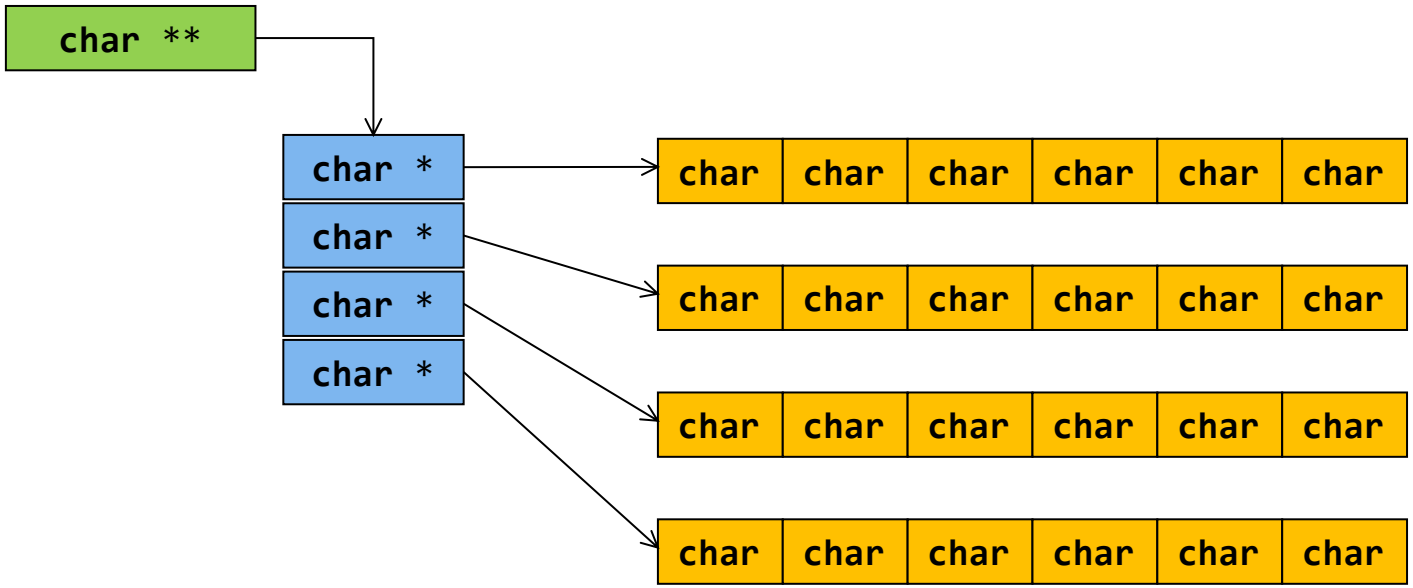
Παράδειγμα

```
#include <stdio.h>
#include <stdlib.h>
#define N 10

main ( ) {

char matrix[N];
char *dynamicdata;
int i;

scanf("%s", matrix);

printf("Hello %s!\n", matrix);

dynamicdata = (char *) malloc( N * sizeof (char));
scanf("%s", dynamicdata);

printf("Hello dynamic %s!\n", dynamicdata);

for (i=0;dynamicdata[i]!=0;i++)
      printf("%c\n", dynamicdata[i]);


}
```

Παράδειγμα

```
#include <stdio.h>
#include <stdlib.h>
#define N 10

main ( ) {
char matrix[N];
char *dynamicdata;
int i, nchars;

scanf("%s", matrix);
printf("Hello %s!\n", matrix);

while (1) {
    printf("How many chars?");
    scanf("%d", &nchars);
    dynamicdata = (char *) malloc( nchars * sizeof (char));
    scanf("%s", dynamicdata);
    printf("Hello dynamic %s!\n", dynamicdata);
    for (i=0;dynamicdata[i]!=0;i++)
           printf("%c\n", dynamicdata[i]);
    free(dynamicdata);
    }
}
```


char **
char *
char *
char *
char *
cha
r



- Δημιούργησε χώρο για τη νέα λέξη
- Δημιούργησε χώρο για τη διεύθυνση της νέας λέξης
- Αποθήκευσε τη διεύθυνση της νέας λέξης

realloc

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define CHARS 10
int main(int argc, char *argv[]) {
    char **mytext = NULL;
    int words = 0;
    char word[CHARS] = "";
    int i;

    while (scanf("%s", word), strcmp(word,"TELOS")) {
            words++;
            mytext = realloc(mytext, words*sizeof(char *));
            mytext[words-1] = malloc (CHARS*sizeof(char));
            strcpy(mytext[words-1], word);
    }

    for (i=0; i<words; i++)
          printf("%s\n", mytext[i]);

    return 0;
}
```

malloc

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define CHARS 10
int main(int argc, char *argv[]) {
 char **mytext = NULL;
 int words = 0;
 char *word;
 int i;

 while (scanf("%s",
word=malloc(CHARS*sizeof(char))),strcmp(word,"TELOS")) {
        words++;
        mytext = realloc(mytext, words*sizeof(char *));
        mytext[words-1] = word;
        }

  for (i=0; i<words; i++)
                printf("%s\n", mytext[i]);

  return 0;
}
```

Δυναμικές δομές δεδομένων

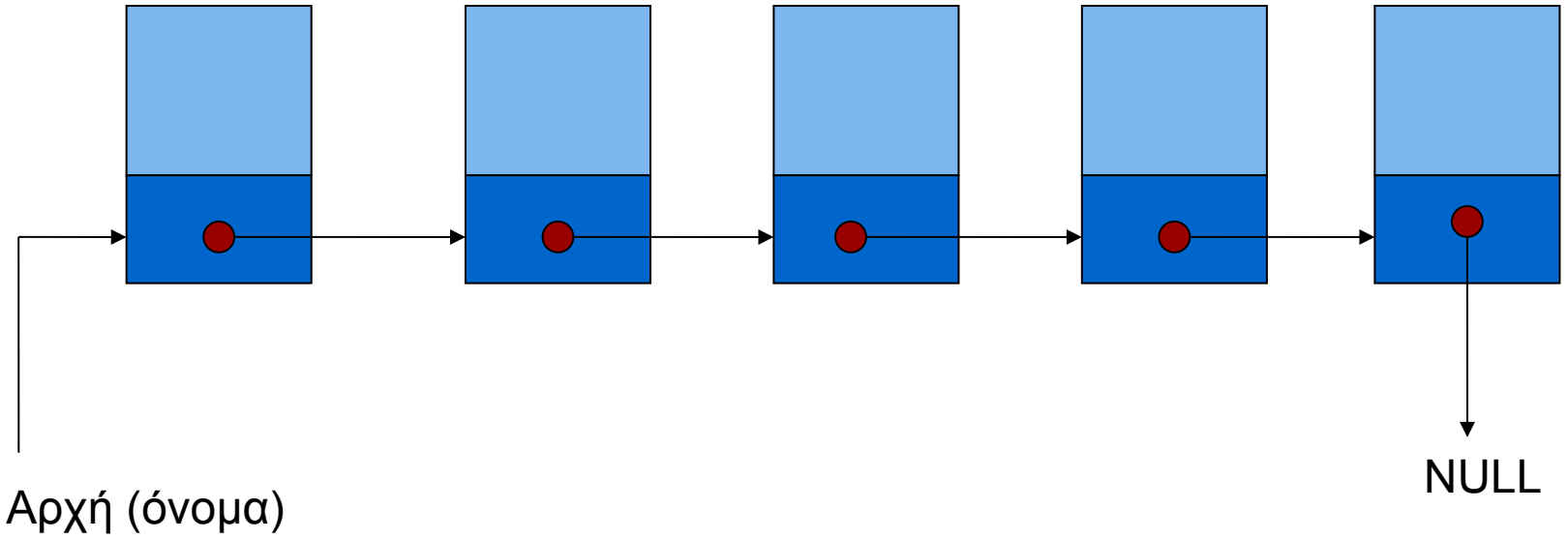
Διασυνδεδεμένες λίστες

Διαχείριση μνήμης (memory management)

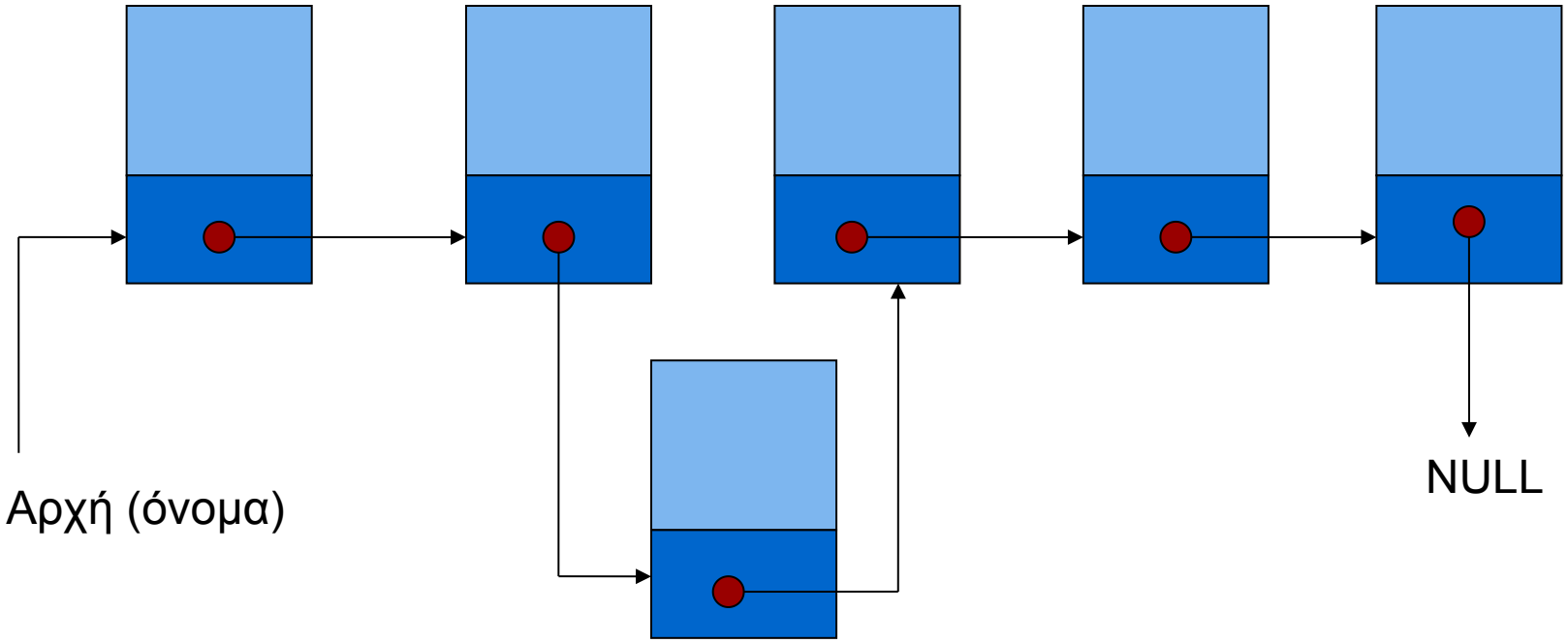
Στατικές και δυναμικές δομές δεδομένων

- **Στατικές**: θέση και μέγεθος καθορίζονται στη μεταγλ ώττιση.
  - `int array[10];`
- **Δυναμικές**: θέση και μέγεθος καθορίζονται κατά την εκτέλεση.
  - Απλούστερος τύπος: η λίστα
- Λίστα: Έτοιμη σε C++/Java, σε C γράφουμε κώδικα

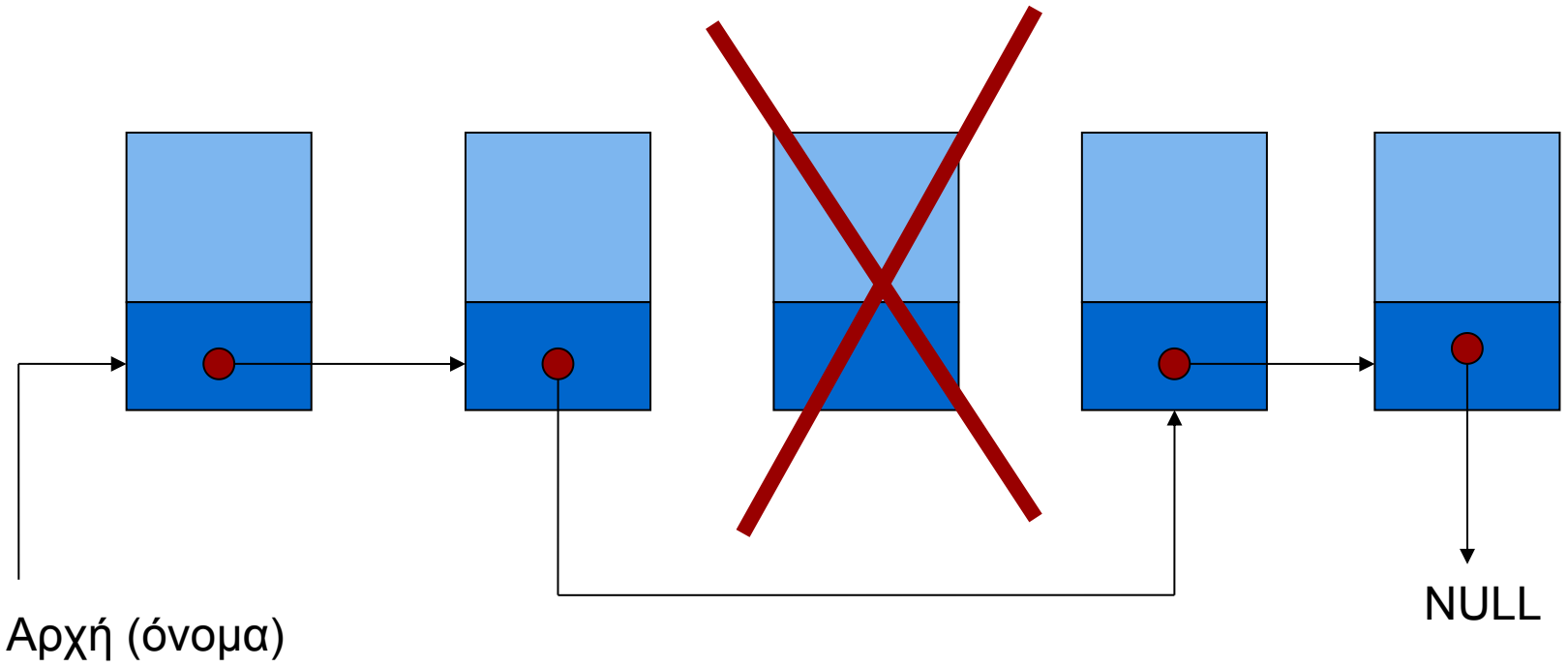
Λίστες και στοιχεία τους



Εισαγωγή στοιχείου



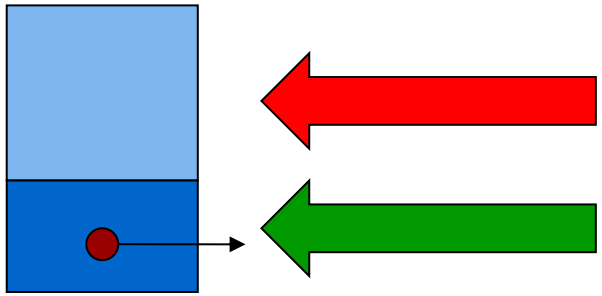
Διαγραφή στοιχείου



Τι γίνεται με τη μνήμη την οποία το στοιχείο καταλάμβανε;

Στοιχείο λίστας

Δεδομένα προβλήματος



Θέση επόμενου στοιχείου

```
#define N 10
struct elem {
    char value[N];
    int count ;
    struct elem *next;
} ;

typedef struct elem Element;
```

Χρήσιμοι συμβολισμοί

```
struct test {
        int a ;
        struct test *next;
        } atest, *atest_ptr, *btest_ptr;
```

Διεύθυνση στοιχείου μετά το atest

atest.next

```
atest.a = 4;
/* δείκτης σε δομή */
atest_ptr = atest.next;
btest_ptr = &atest;
(*btest_ptr).a ++;
/* το ίδιο με το προηγούμενο */
(btest_ptr->a)++ ;
```

Αναφορά σε πολλά στοιχεία

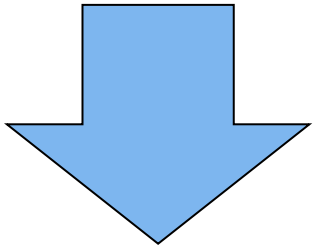
- Με στατικό τρόπο
- `struct test atest, btest, ctest ;`
- `struct test manytests[10] ;`
- Ή δυναμικά…
  - Πώς μπορεί να δημιουργεί «μεταβλητές» την ώρα της εκτέλεσης;
  - Βασική ιδέα:
    - δε δημιουργεί νέα ονόματα μεταβλητών,
    - Αλλά δεσμεύει αναλυτικά κατάλληλες περιοχές μνήμης και κρατάει αναφορές σε αυτές

Όχι πραγματική λίστα, ενδιάμεσο βήμα

```
#include <stdio.h>
#define L 30
struct listelement {
            char name[L];
            int age;
            struct listelement *next;
};
typedef struct listelement Listelement;
int main(int argc, char *argv[]) {
          /* Initialize */
          Listelement c = { "Giannis", 5, NULL},
                                b = { "Maria", 10, NULL },
                                a = { "Vassilis", 3, NULL};
          Listelement *iterator;
          /* Connect to list*/
          a.next = &b;
          b.next = &c;
          /* Access elements */
          for (iterator = &a; iterator != NULL ; iterator = (*iterator).next)
          {
                    printf( "name: %s\n", (*iterator).name);
                    printf("age: %d\n", (*iterator).age);
           }
          return 0;
}
```

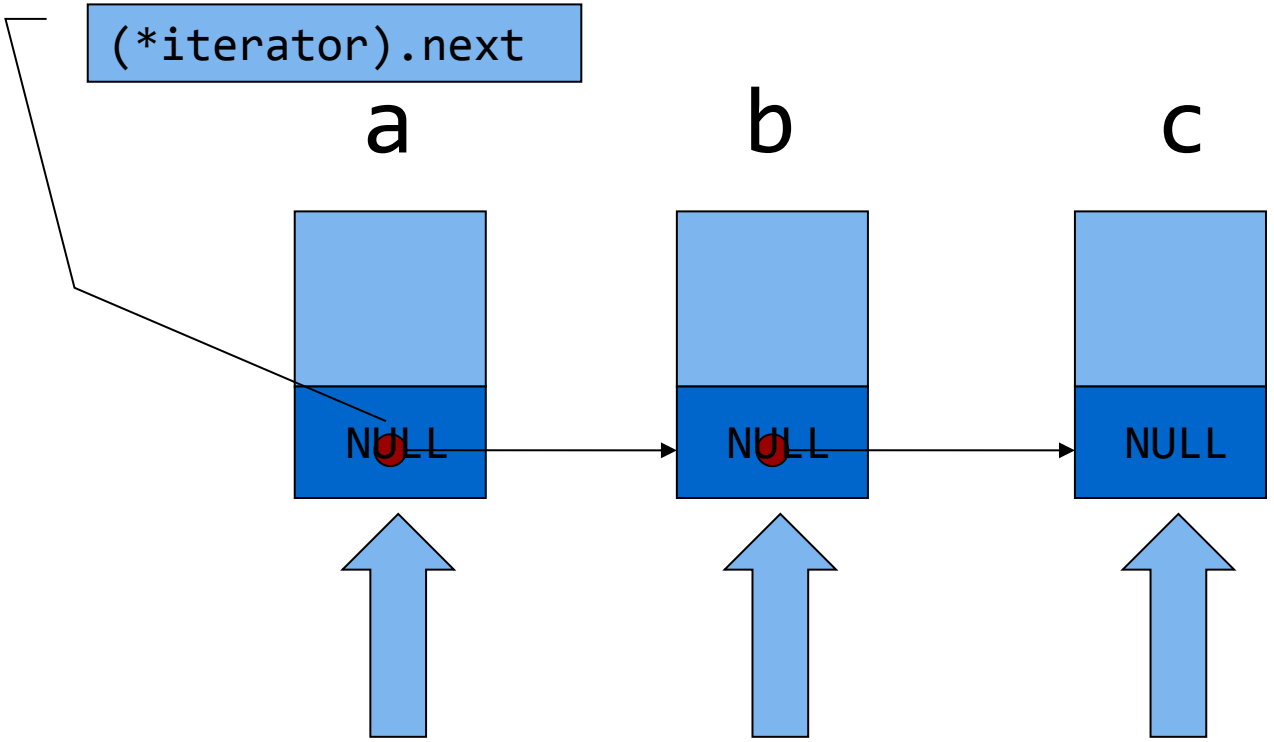
Απλοποιημένος συμβολισμός

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



- `test->next`

```
Listelement c = { "Giannis", 5, NULL},
                b = { "Maria", 10, NULL },
                a = { "Vassilis", 3, NULL};

Listelement *iterator;
```




```
a.next = &b;
b.next = &c;
```

```
iterator = &a  iterator = &b   iterator = &c

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

Λίστα

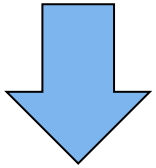
- Δεν χρησιμοποιήθηκαν τα ονόματα a, b, c για να επεξεργαστούμε τη λίστα!

- Αρκεί η διεύθυνση του πρώτου στοιχείου μόνο!
  - Η διεύθυνση του επόμενου στοιχείου είναι `iterator->next`

Λίστα

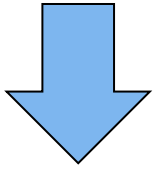
Στατική δήλωση

```
Listelement c = { "Giannis", 5, NULL};
```



Δυναμική δήλωση στοιχείου της λίστας

```
Listelement *element_ptr;
element_ptr = (Listelement *) malloc ( sizeof (Listelement));
strcpy((*element_ptr).name,"Ntina");
(*element_ptr).age = 9;
(*element_ptr).next = NULL;
```



Άλλος συμβολισμός, ενδεικτικά:
element_ptr->name αντί (*element_ptr).name

```
Listelement *element_ptr;
element_ptr = (Listelement *) malloc ( sizeof (Listelement));
strcpy(element_ptr->name,"Ntina");
element_ptr->age = 9;
element_ptr->next = NULL;
```

Παράδειγμα

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define L 30
struct listelement {
                char name[L];
                int age;
                struct listelement *next;
};
typedef struct listelement Listelement;
typedef Listelement * Listelement_ptr;
int main(int argc, char *argv[]) {
             /* Initialize */
             Listelement_ptr iterator=NULL, element_ptr,  previous_ptr, first_ptr;          /* create elements and
put them to list*/
              element_ptr = (Listelement_ptr) malloc ( sizeof (Listelement));
              strcpy(element_ptr->name,"Ntina");
              element_ptr -> age = 9;
              element_ptr -> next = NULL;
              first_ptr = element_ptr;
              previous_ptr = element_ptr;
              element_ptr = (Listelement_ptr) malloc ( sizeof (Listelement));
              strcpy(element_ptr->name,"Giannis");
              element_ptr -> age = 6 ;
              previous_ptr -> next = element_ptr;
              /* Access elements */
              for (iterator = first_ptr; iterator != NULL ; iterator = iterator->next) {
                            printf( "name: %s\n", iterator->name);
                            printf("age: %d\n", iterator->age);
               }
              return 0;
}
```

mytypes.h

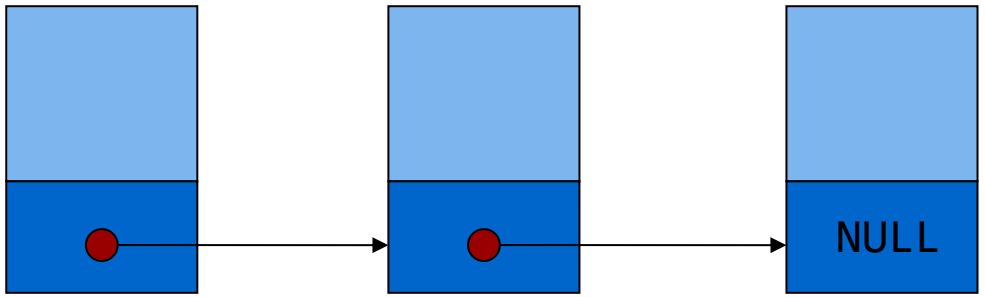
```
#define L 30
struct listelement {
        char name[L];
        int age;
        struct listelement *next;
};
typedef struct listelement Listelement;
typedef Listelement * Listelement_ptr;
typedef Listelement * List;
```

Δημιουργία στοιχείου

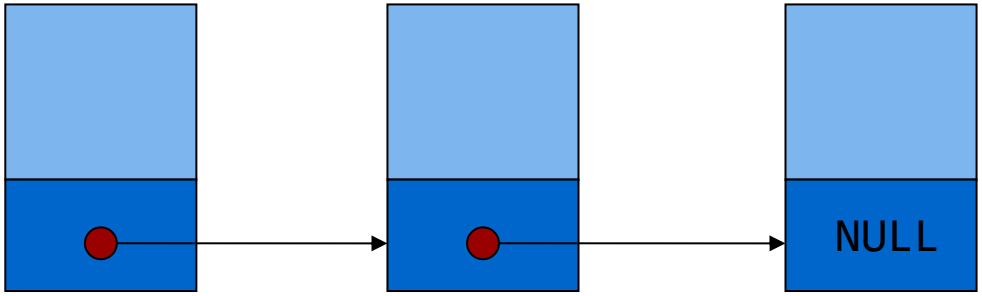
```
#include <string.h>
#include <stdlib.h>
#include "mytypes.h"

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

        newelement_ptr = (Listelement_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;
          }


}
```

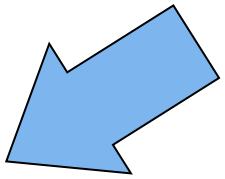
Όταν βγει από το βρόχο, ο iterator δείχνει στο τελευταίο στοιχείο

Παράδειγμα

```
#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);
void deleteelement(List *, Listelement_ptr);
int main(int argc, char *argv[]) {
              /* 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);
                              deleteelement(&nameslist, element_ptr);
                              reportlist(nameslist);
                 }
   } while (nameslist!=NULL) ;
 return 0;
}
```

Παράδειγμα

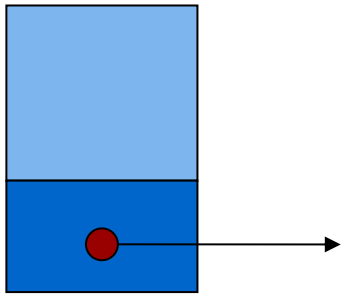
Παράμετρος: δείκτης σε λίστα ή διπλός δείκτης σε στοιχείο λίστας



```
#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;
}
```

Λίστες στη C

```
#define N 10
struct elem {
 char value[N];
 int count ;
 struct elem *next;
} ;

typedef struct elem Element;
```



Τα στοιχεία της λίστας είναι τύπου Element

```
int FindWord(const char *, Element *);
void AppendWord(const char *, Element **);
void DeleteWord(const char *, Element **);
void ReportList(Element * const);
Element *CreateNode (void );
```

Υπάρχουν διαδικασίες που μπορεί να αλλάξουν την αρχική διεύθυνση (διαγραφή πρώτου στοιχείου, προσθήκη πρώτου στοιχείου)

Δέσμευση μνήμης για στοιχείο λίστας

```
Element *CreateNode (void ) {
 return (Element *) malloc (sizeof (Element)) ;
}
```

Προσθήκη στοιχείου στη λίστα

```
void AppendWord(const char *word, Element **list_ptr) {
Element *item_iterator = *list_ptr;
Element *new_word = CreateNode();

if (*list_ptr!=NULL) {  Αν η λίστα υπάρχει, να βάλεις το νέο στοιχείο στο τέλος…
        for( ; item_iterator->next!=NULL;item_iterator=item_iterator->next) ;
        item_iterator->next = new_word;
   }
else
   {    Αν η λίστα δεν υπάρχει, να είναι αυτό το πρώτο στοιχείο…
        *list_ptr = new_word;
   }

   strcpy(new_word->value, word);
   new_word->next=NULL;
}
```

Παράδειγμα

```
void ReportList(Element * const list) {
Element *item_iterator = list;

if (list == NULL) {
    printf("empty list\n");
    }
else
    {
    for(;item_iterator!=NULL;item_iterator=item_iterator->next)
          printf("word: %s\n", item_iterator->value);
    }

}
```

Αναζήτηση στη λίστα

```
int FindWord(const char *s, Element *alist) {
 Element *test = alist;

 int found = 0;

 while ((test->next != NULL) &&(!found)) {
   printf("%s ", test->value);
   found = !strcmp (s, test->value);
   if (found) (test->count)++;
   test = test->next;
 }

 return found;
}
```

Διαγραφή στοιχείου/Αποδέσμευση μνήμης

```
void DeleteWord(const char *word, Element **list_ptr) {
 Element *test = *list_ptr, *prev;
 int found = 0;

 prev = NULL;
 printf("Deleting...\n");
 while ((test != NULL) &&(!found)) {
   printf("%s ", test->value);
   found = !strcmp (word, test->value);
   if (found)
           (test->count)++;
   else {
           prev = test;
    test = test->next;
           }
 }
 if (found) {
    printf("word found: %s.\n", test->value);
    if (prev == NULL)
           *list_ptr = test->next;
    else {
           printf("previous word: %s\n", prev->value);
           prev->next = test->next;
           }
    free(test);
    }
}
```

Σημείωμα αναφοράς

- Copyright Πανεπιστήμιο Πατρών, Παλιουράς Βασίλειος , Δερματάς Ευάγγελος «Αρχές Προγραμματισμού ». Έκδοση: 1.0. Πάτρα 2015
- Διαθέσιμο από τη δικτυακιακή διέυθυνση https://eclass.upatras.gr/modules/