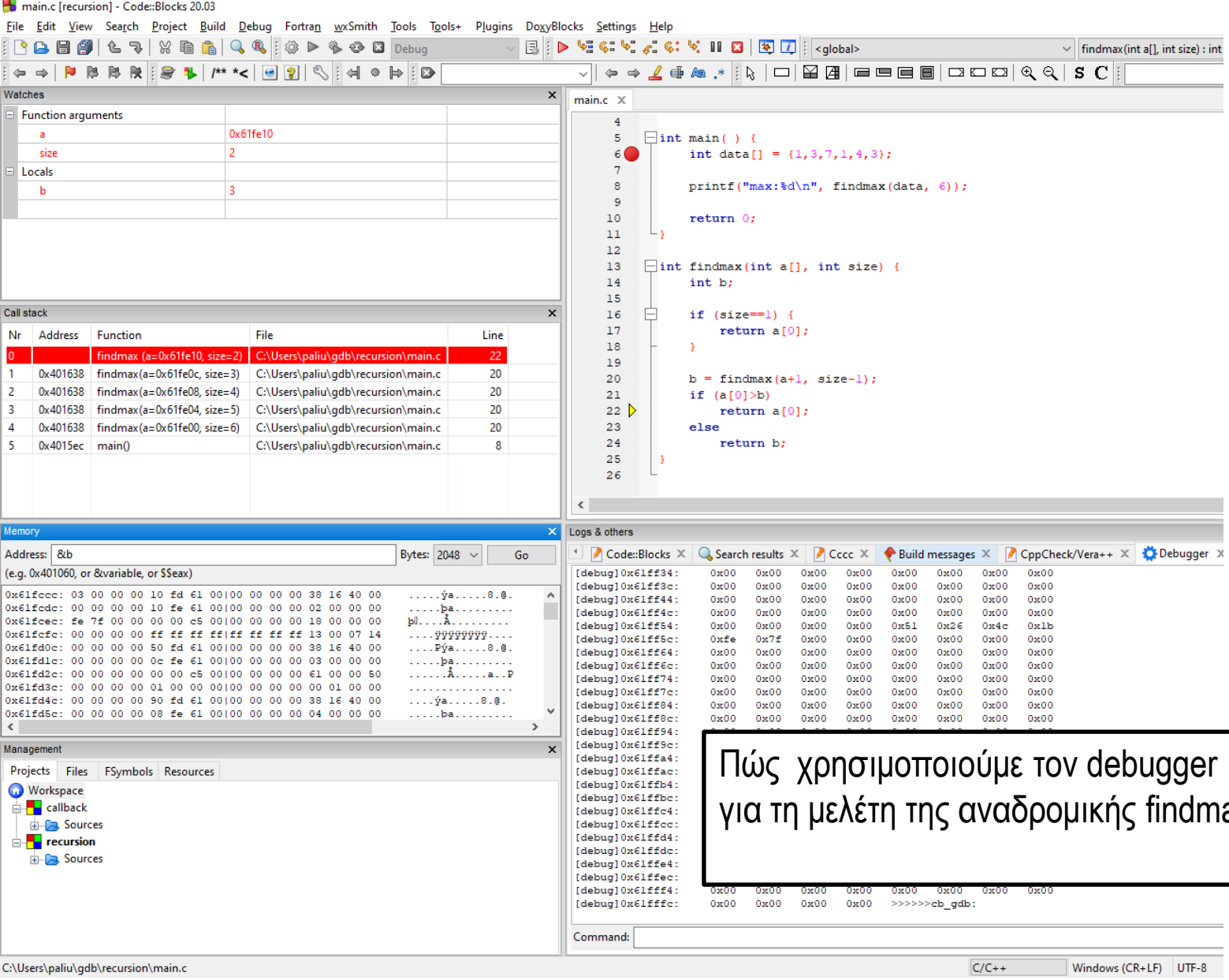
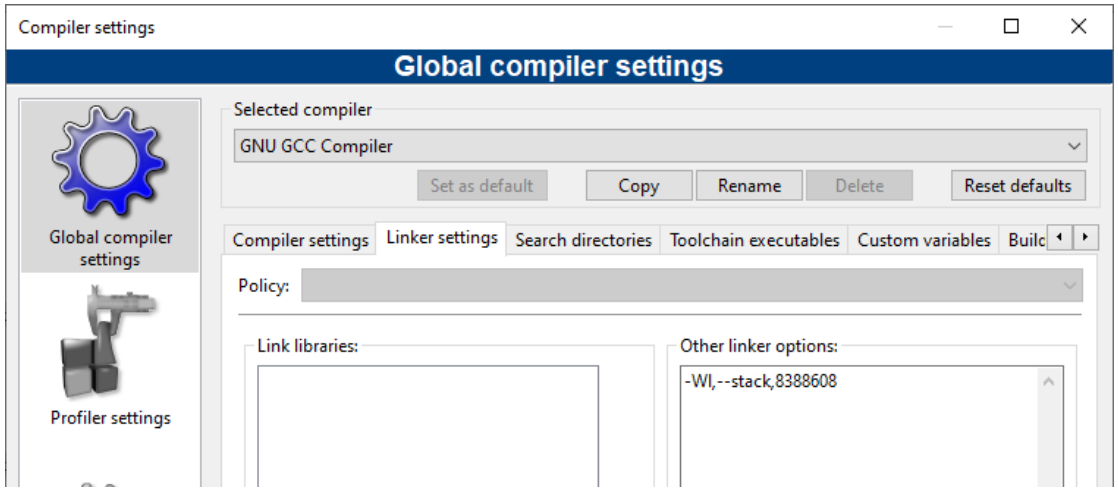
Διαδικαστικός Προγραμματισμός

Βασίλης Παλιουράς
paliuras@ece.upatras.gr


main.c [recursion] - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Watches
Function arguments
a 0x61fe10
size 2
Locals
b 3
Call stack
Nr Address Function File Line
0 findmax (a=0x61fe10, size=2) C:\Users\paliu\gdb\recursion\main.c 22
1 0x401638 findmax(a=0x61fe0c, size=3) C:\Users\paliu\gdb\recursion\main.c 20
2 0x401638 findmax(a=0x61fe08, size=4) C:\Users\paliu\gdb\recursion\main.c 20
3 0x401638 findmax(a=0x61fe04, size=5) C:\Users\paliu\gdb\recursion\main.c 20
4 0x401638 findmax(a=0x61fe00, size=6) C:\Users\paliu\gdb\recursion\main.c 20
5 0x4015ec main() C:\Users\paliu\gdb\recursion\main.c 8
Memory
Address: &b
Bytes: 2048
Go
(e.g. 0x401060, or &variable, or $$eax)
Management
Projects Files FSymbols Resources
Workspace
callback
Sources
recursion
Sources
main.c
4
5 int main( ) {
6 int data[] = {1,3,7,1,4,3};
7
8 printf("max:%d\n", findmax(data, 6));
9
10 return 0;
11 }
12
13 int findmax(int a[], int size) {
14 int b;
15
16 if (size==1) {
17 return a[0];
18 }
19
20 b = findmax(a+1, size-1);
21 if (a[0]>b)
22 return a[0];
23 else
24 return b;
25 }
26
Logs & others
Code::Blocks Search results Cccc Build messages CppCheck/Vera++ Debugger
Command:
C:\Users\paliu\gdb\recursion\main.c
C/C++ Windows (CR+LF) UTF-8
Πώς χρησιμοποιούμε τον debugger
για τη μελέτη της αναδρομικής findmax




```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
char * readenglish(char []);
void printenglish(char *) ;
#define SIZE 638977
int main(void)
{
    char * d_ptr ;
    char dictionary[SIZE];
    d_ptr = readenglish(dictionary) ;
    printenglish(d_ptr) ;

    return 0;
}

char * readenglish(char dictionary[] )  {

    FILE *fp ;
    fp = fopen("englishWords.txt","rb");

    if (fp!=NULL) {
        fread(dictionary, 1, 638976, fp);
        fclose(fp);
    }

    dictionary[638976] = 0;

    return dictionary ;
}
```

```c
void printenglish(char * d ) {
    char word[100] ;
    while (sscanf(d, "%99s", word)!=EOF) {
            printf("%s\n", word);
            d += strlen(word)+1 ;
    }

    return ;
}
```

- Δυναμικές δομές δεδομένων
- Διασυνδεδεμένες λίστες
- Διαχείριση μνήμης (memory management)

By value/by reference

```
{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;
  }
```

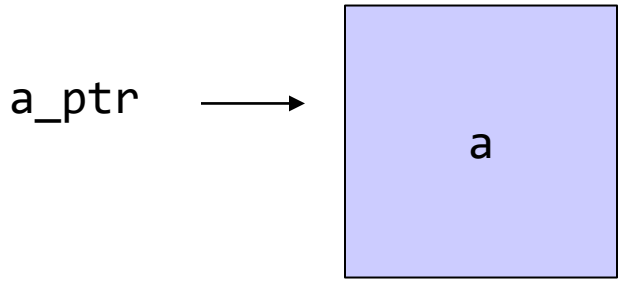
Τ οποιοσδήποτε τύπος

Παράδειγμα

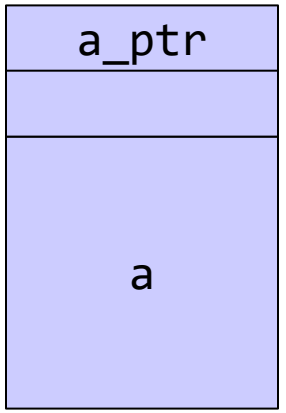
```
typedef char **  T;
```

Review δείκτες ως αναφορές

```
T a;

T * a_ptr;

a_ptr = &a;
```



*a_ptr ⟺ a

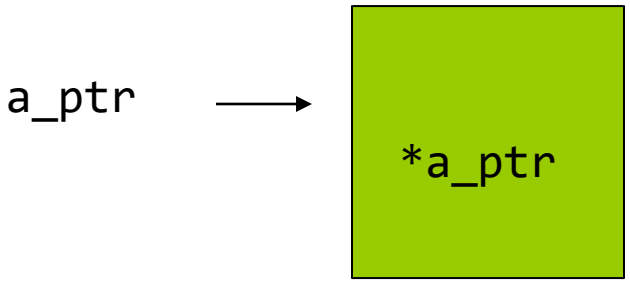


a, a_ptr στο stack

Θεμελιώδης τεχνική: αναφορές σε δεδομένα στο heap

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

`a_ptr,` στο stack, αλλά τα δεδομένα στο heap

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

typedef struct data {
    char word[9+1] ;
    int code; } Data ;

Data create (void) ;
int print(Data) ;

int main(void)
{
    Data a;

    a = create();
    print(a);

    return 0;
}
```

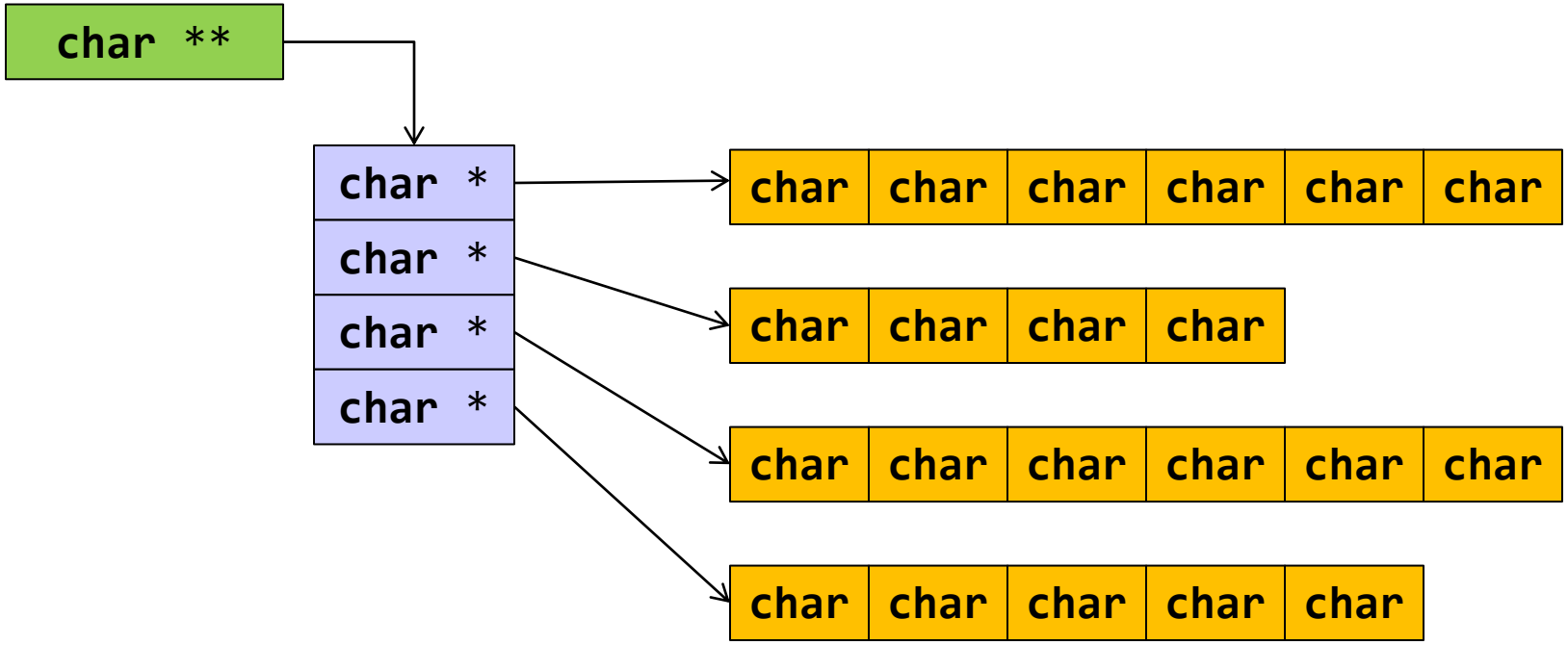
```c
Data create(void) {
    Data temp;

    temp.code = 123;
    strcpy(temp.word, "hey you");

    return temp;
}

int print(Data x) {

    printf("%s %d\n", x.word, x.code) ;

    return 0;
}
```

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

typedef struct data {
    char word[9+1] ;
    int code;
    } Data ;
typedef  Data * Data_ptr ;

Data * create (void) ;
int print(Data) ;

int main()
{
    Data_ptr a_ptr ;
    Data_ptr b_ptr ;
    Data_ptr c_ptr ;
    Data_ptr * data_arr;
    int i ;

    data_arr = malloc(10 * sizeof(Data_ptr) );

    for (i=0;i<10;i++) {
        data_arr[i] = create();
        print(* (data_arr[i]) );
    }

    b_ptr = create();
    print(*b_ptr);

    c_ptr = create();
    print(*c_ptr);

    for (i=0;i<10;i++) {
        print(* (data_arr[i]) );
    }

    return 0;
}
```

```c
Data * create(void) {
    Data * temp;
    static int i = 0;
    temp = malloc(sizeof (Data) );

    /*(*temp).code = i++; */
    temp -> code = i ++;

    /*strcpy((*temp).word, "hey you"); */
    strcpy( temp->word, "hey you");
    printf("%p\n", temp);
    return temp;
}

int print(Data x) {

    printf("%s %d\n", x.word, x.code) ;

    return 0;
}
```

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

typedef struct data {
    char word[9+1] ;
    int code;
    struct data * next ;
    } Data ;
typedef  Data * Data_ptr ;

Data_ptr create (void) ;
void report (Data_ptr) ;

int print(Data) ;

int main()
{
    Data_ptr a_ptr ;
    Data_ptr b_ptr ;
    Data_ptr c_ptr ;
    Data_ptr * data_arr;
    int i ;

    data_arr = malloc(10 * sizeof(Data_ptr) );

    for (i=0;i<10;i++) {
        data_arr[i] = create();
        print(* (data_arr[i]) );
    }


   for (i=0;i<9;i++) {
        (*(data_arr[i])).next  = data_arr[i+1] ;
    }

    b_ptr = create();
    print(*b_ptr);

    c_ptr = create();
    print(*c_ptr);

    for (i=0;i<10;i++) {
        print(* (data_arr[i]) );
    }

    printf("recursive print\n");
    report (data_arr[0]) ;

    return 0;
}
```

```c
Data_ptr create(void) {
    Data_ptr temp;
    static int i = 0;
    temp = malloc(sizeof (Data) );


    temp -> code = i ++;
    strcpy( temp->word, "hey you");
    temp -> next = NULL;
    printf("%p\n", temp);
    return temp;
}

int print(Data x) {

    printf("%s %d\n", x.word, x.code) ;

    return 0;
}

void report(Data_ptr x_ptr) {

    if (x_ptr == NULL) return ;

    print(*x_ptr );

    report(x_ptr -> next) ;

    return ;
}
```


char **
char *
char *
char *
char *
char char char char char char
char char char char
char char char char char char
char char char char char

Στατικές και δυναμικές δομές δεδομένων

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

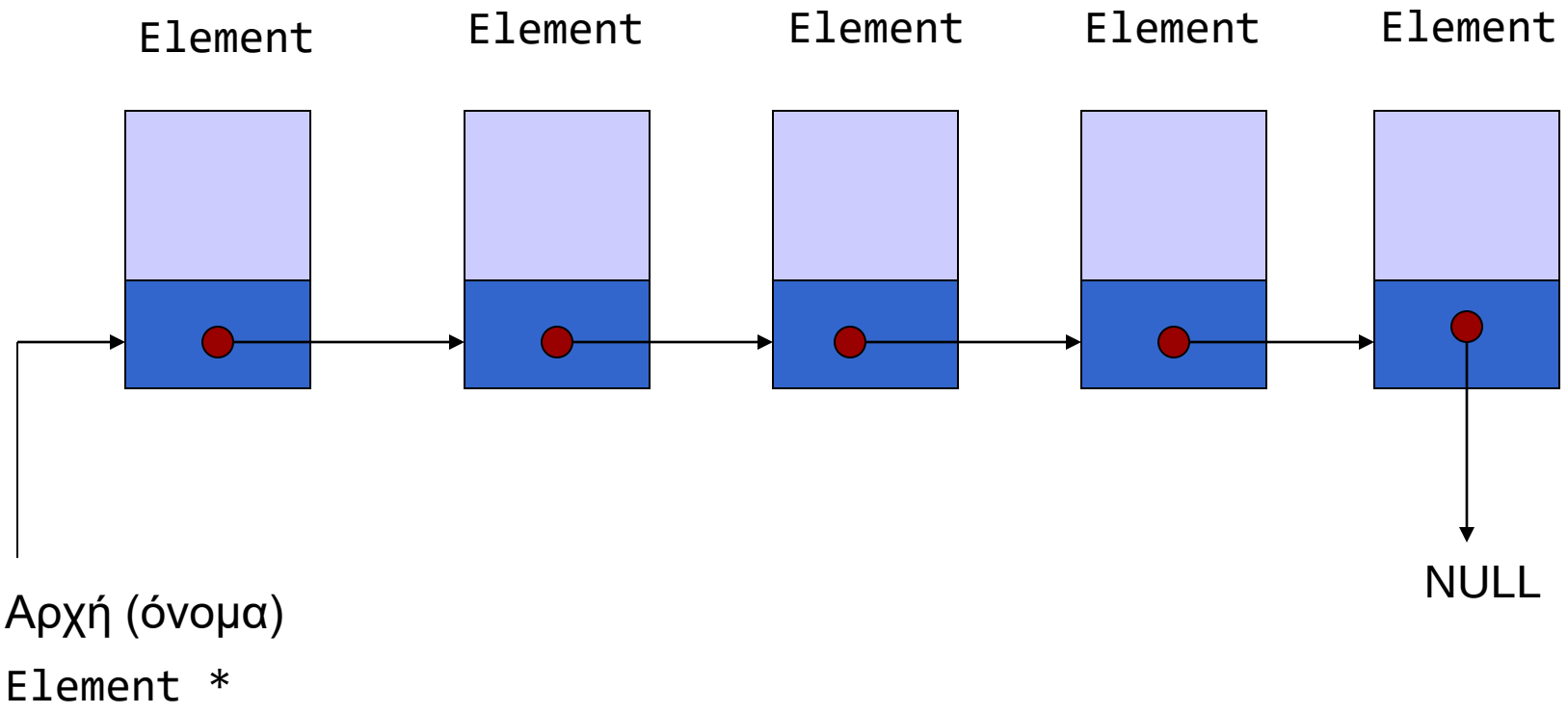
Παρεμβολή στη θέση 3 σε πίνακα σταθερού μεγέθους

0 aa
1 bb
2 cc
3 dd
4 ee
5 ff
6 gg
7 hh

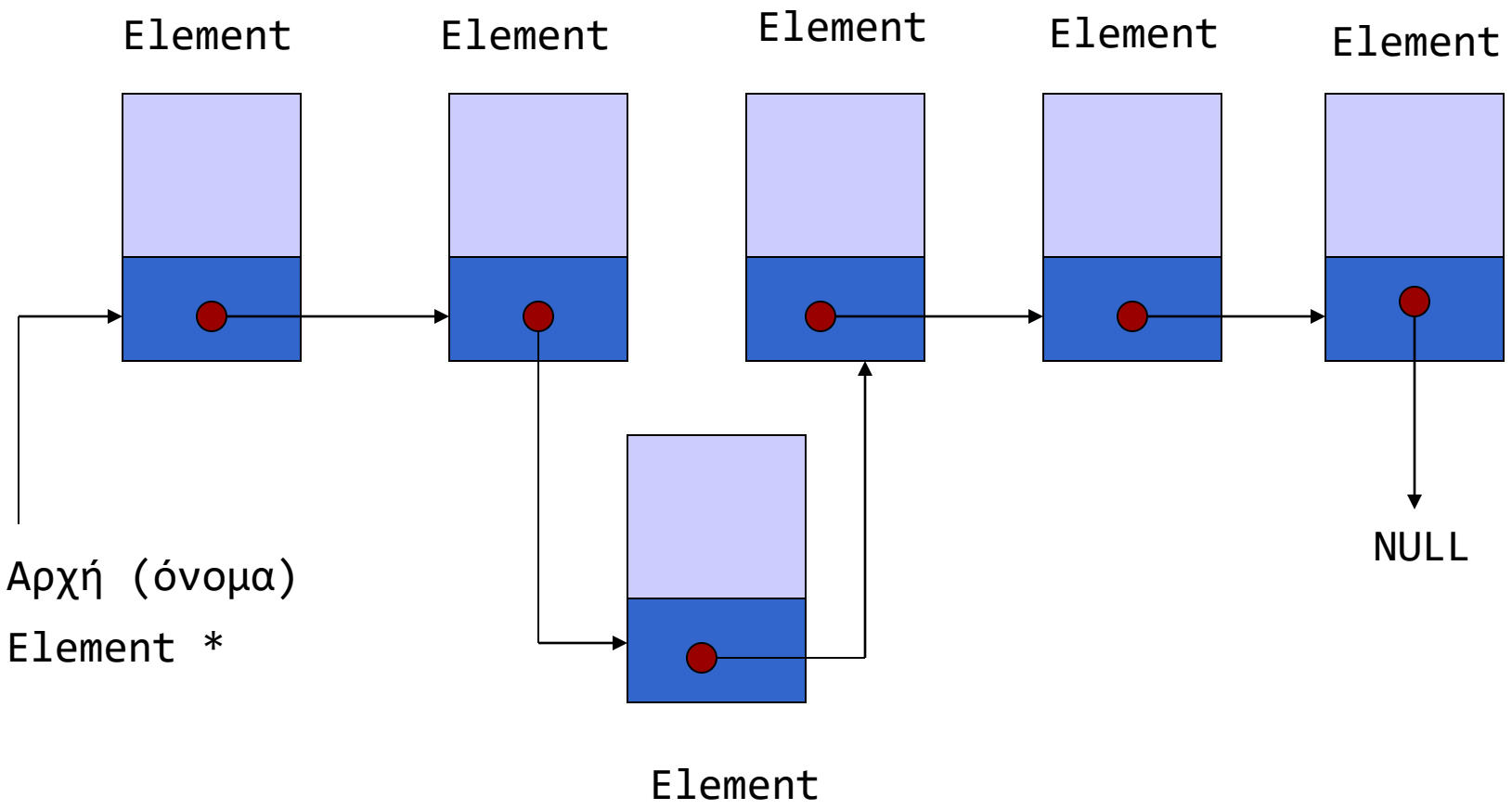
0 aa
1 bb
2 cc
3
4 dd
5 ee
6 ff
7 gg

0 aa
1 bb
2 cc
3 **ZZ**
4 dd
5 ee
6 ff
7 gg

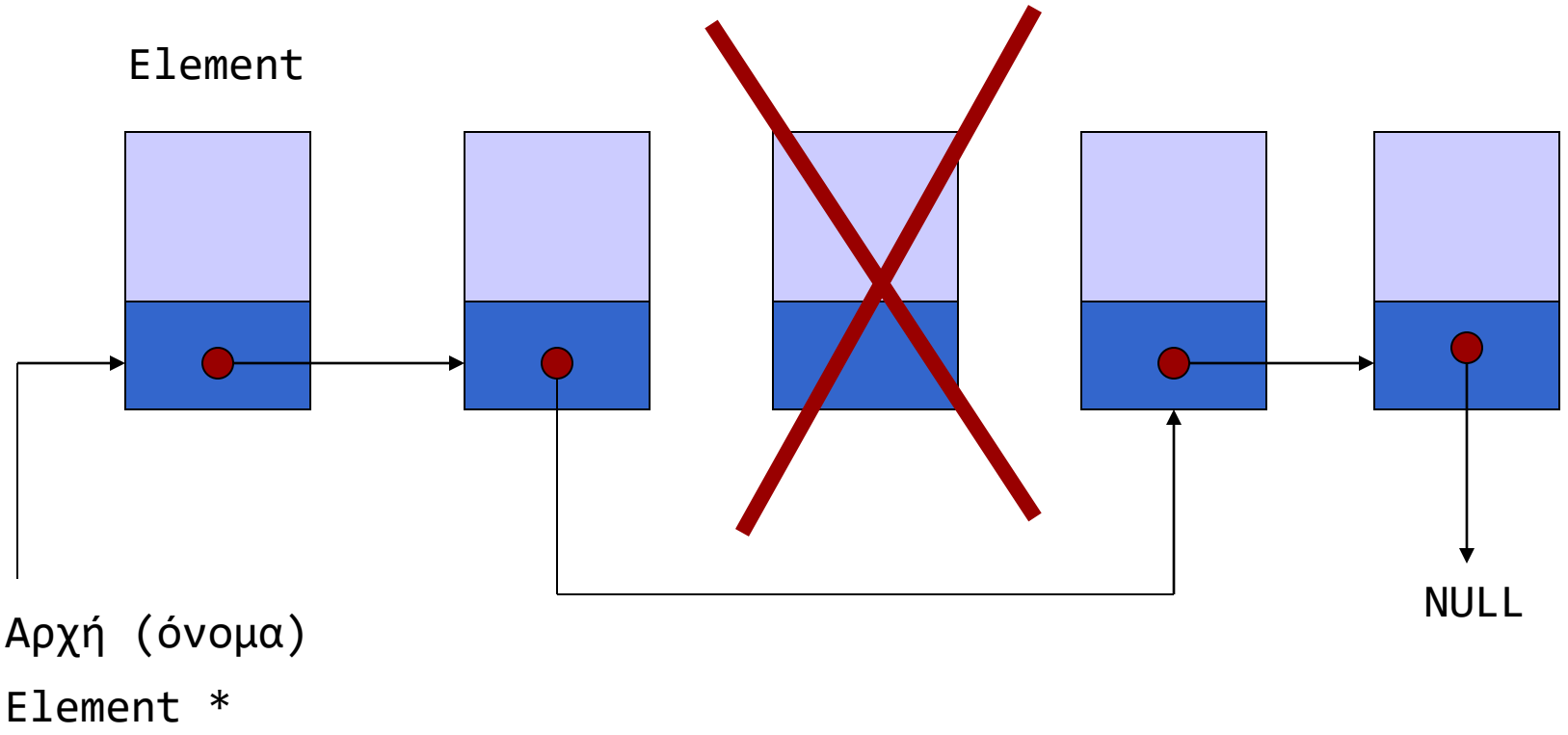
Λίστες και στοιχεία τους



Εισαγωγή στοιχείου



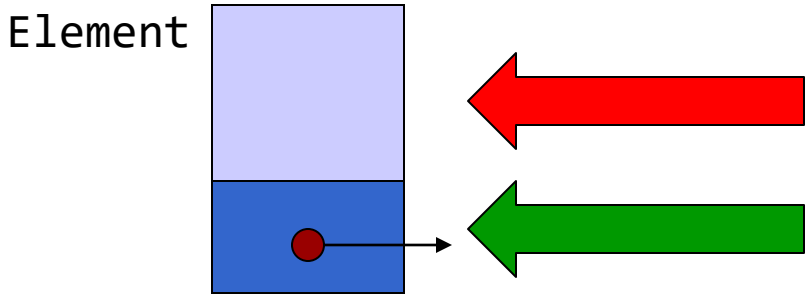
Διαγραφή στοιχείου



Τι γίνεται με τη μνήμη την οποία το στοιχείο καταλάμβανε;

Στοιχείο λίστας

Δεδομένα προβλήματος




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

typedef struct elem Element;
```

Θέση επόμενου στοιχείου

Χρήσιμοι συμβολισμοί

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

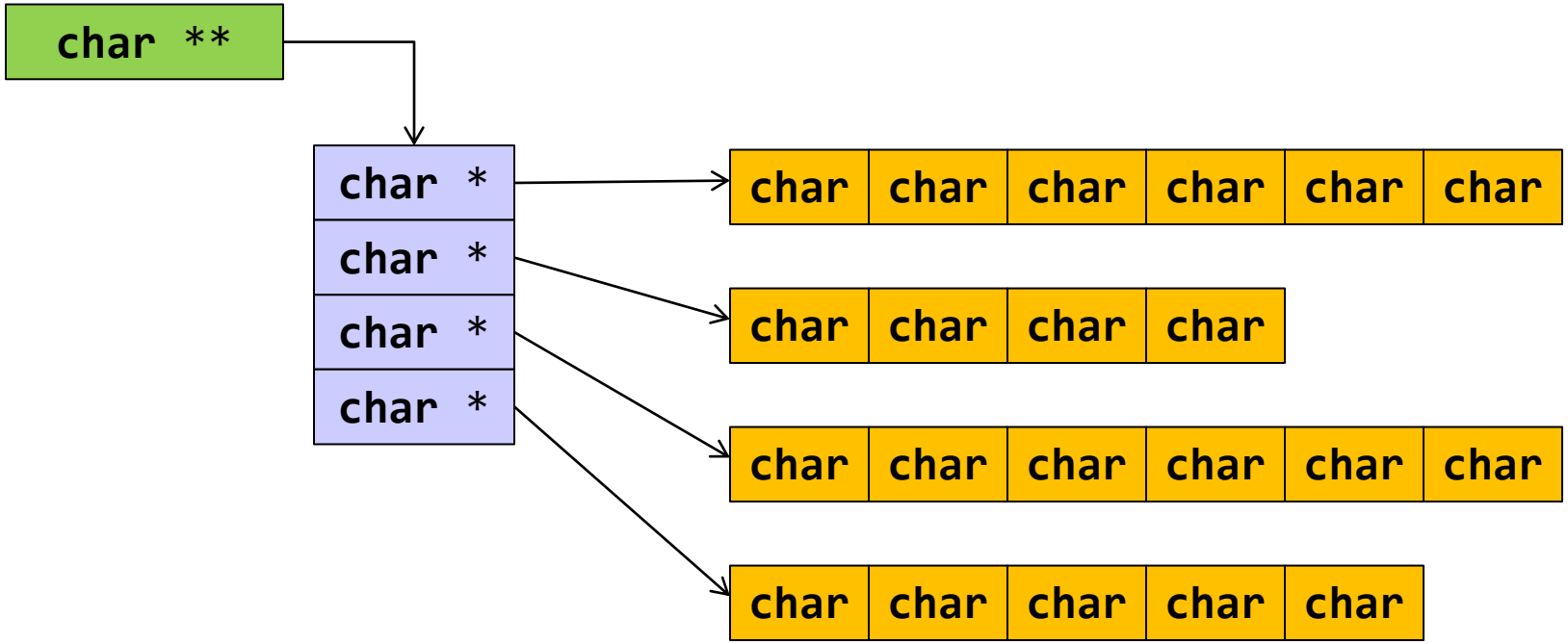
Η διεύθυνση του στοιχείου που βρίσκεται μετά το atest, αποθηκεύεται στο atest.next

```
/* ακέραιος */
atest.a = 4;
/* δείκτης σε δομή */
atest_ptr = atest.next;
btest_ptr = &atest;
(*btest_ptr).a ++;
/* άλλος τρόπος, ισοδύναμος */
(btest_ptr->a)++ ;
```

Αναφορά σε πολλά στοιχεία

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

Έχουμε ήδη χρησιμοποιήσει την ιδέα…

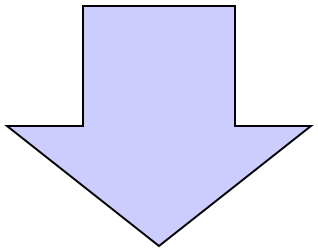


Όχι πραγματική λίστα, ενδιάμεσο βήμα

```c
#include <stdio.h>
#define L 30
struct listelement {
            char name[L];
            int age;
            struct listelement *next;
};
typedef struct listelement Listelement;
int main(void) {
        /* 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;
}
```

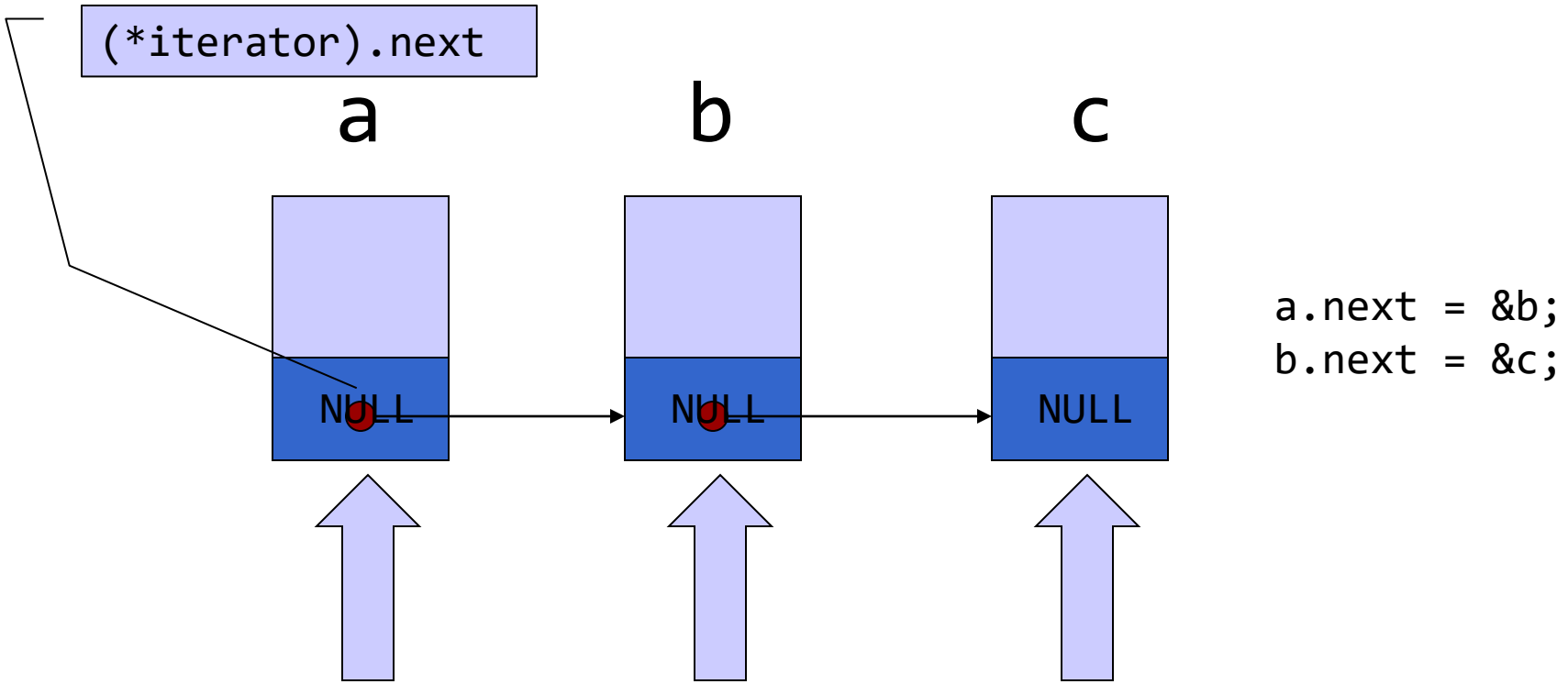
Απλοποιημένος συμβολισμός

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



- `test_ptr->next`

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

Listelement *iterator;
```




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

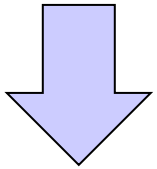
```
iterator = &a  iterator = &b  iterator = &c

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

- 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`

Στατική δήλωση

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

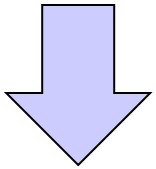


Δυναμική δημιουργία στοιχείου της λίστας

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

Δυναμική δημιουργία στοιχείου της λίστας

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

```c
#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(void) {
          /* 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;
```

Τύποι λίστας στο mytypes.h

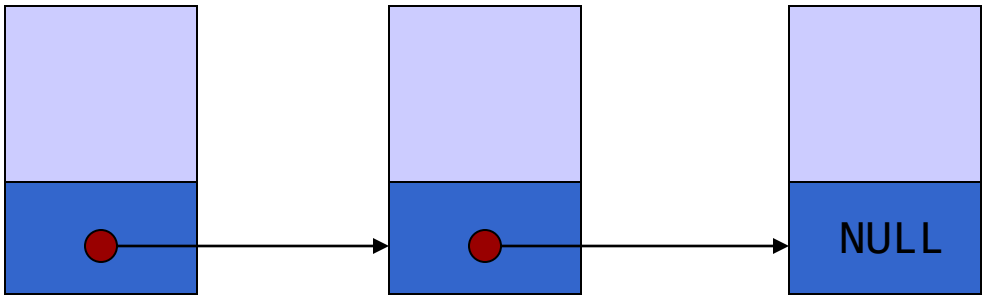
- Listelement
  - Στοιχείο λίστας (**struct** listelement)
- Listelement_ptr
  - Δείκτης σε στοιχείο λίστας. Ίδιο με
    - **struct** listelement *
    - Listelement *
- List
  - Δείκτης σε στοιχείο λίστας
  - Μεταβλητές αυτού του τύπου δείχνουν
    - Στο πρώτο στοιχείο της λίστας ή
    - Έχουν την τιμή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

Δημιουργία στοιχείου

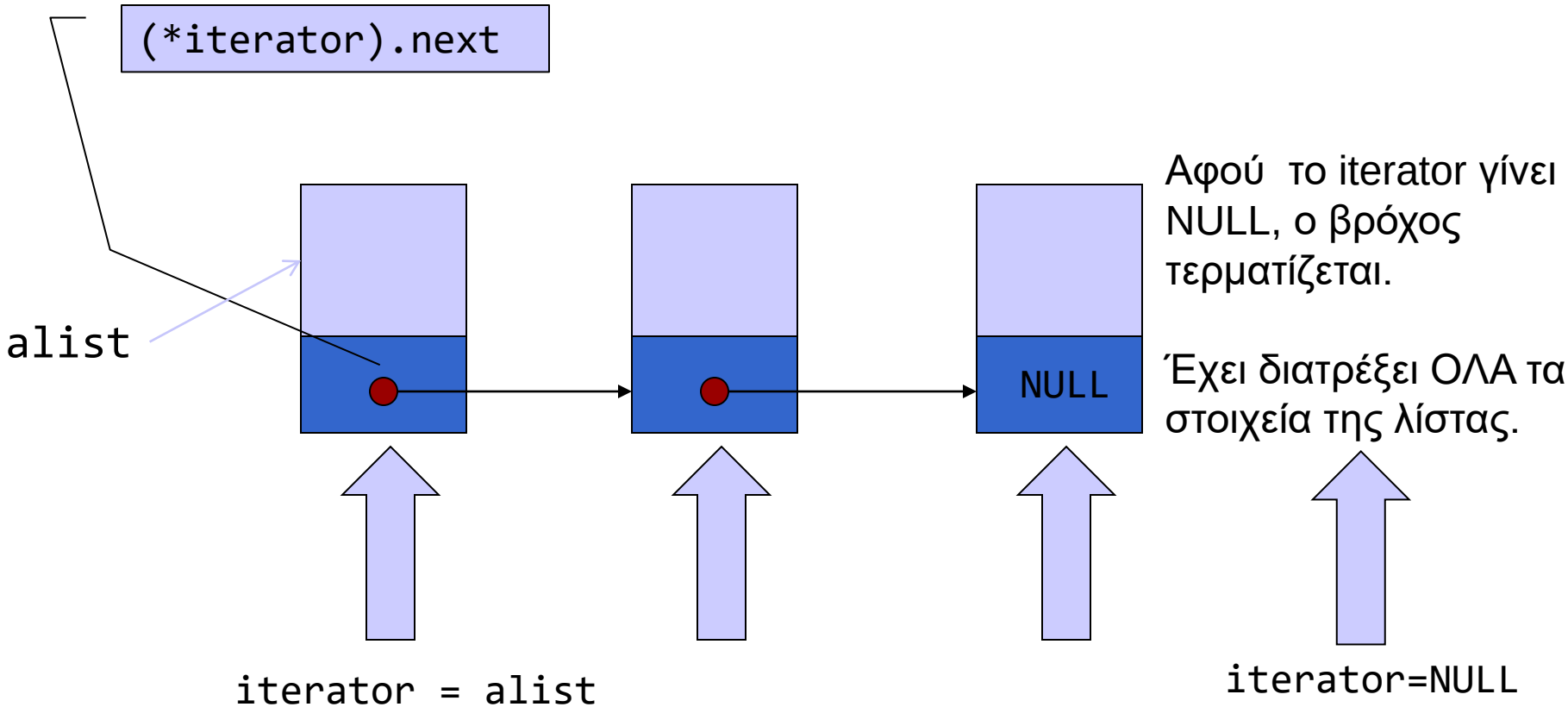
```
#include <string.h>
#include <stdlib.h>
#include "mytypes.h"

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

        newelement_ptr = (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);
         }
}
```

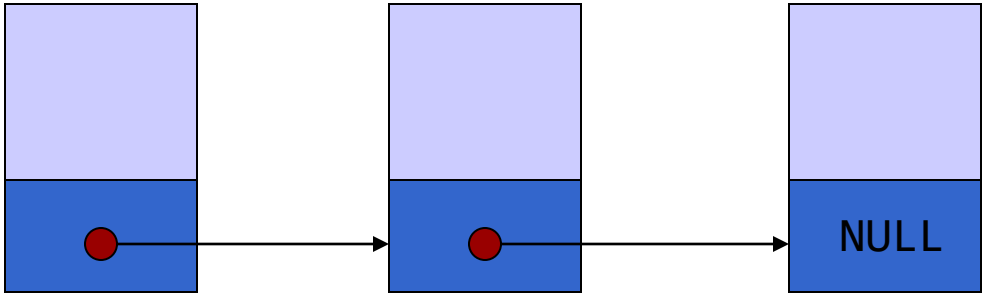


```
Listelement_ptr iterator = alist;
```




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

Προσθήκη στοιχείου στο τέλος λίστας

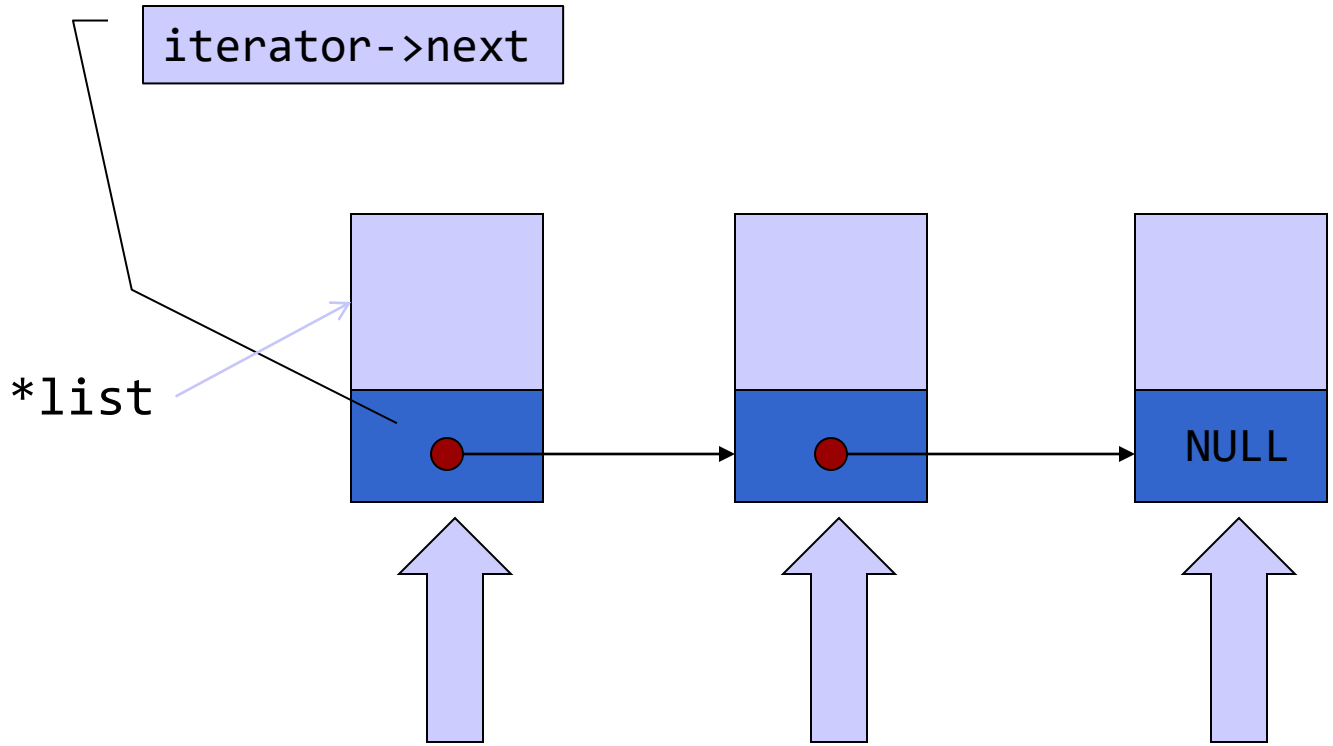


Παράμετρος δείκτης σε λίστα ή διπλός δείκτης σε στοιχείο

```
#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

```
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);
void 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);
                       deleteelement(&nameslist, element_ptr);
                       reportlist(nameslist);
                    }
          } while (nameslist!=NULL) ;
 return 0;
}
```

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 0;
 }
```

```
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;
 }
```

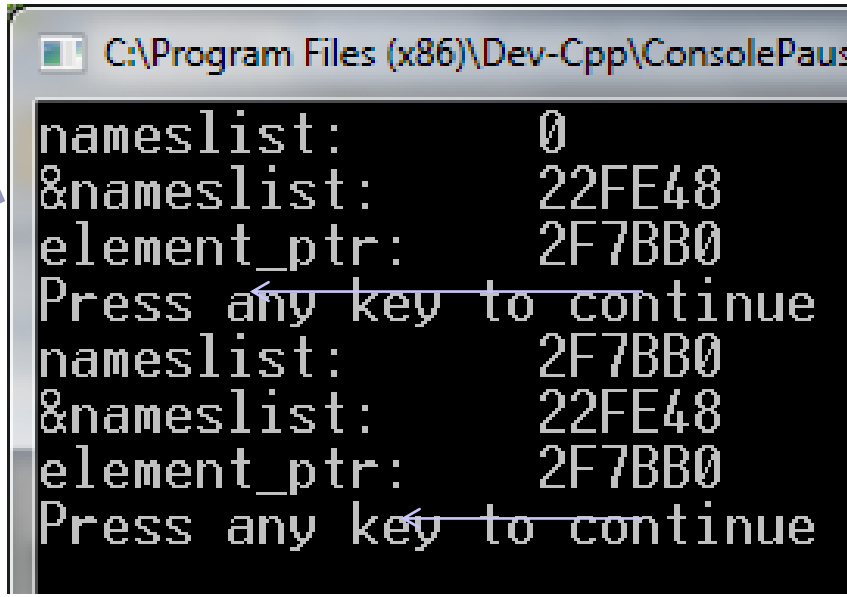
```
void report(List lst) {
           Node * iter;

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

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)

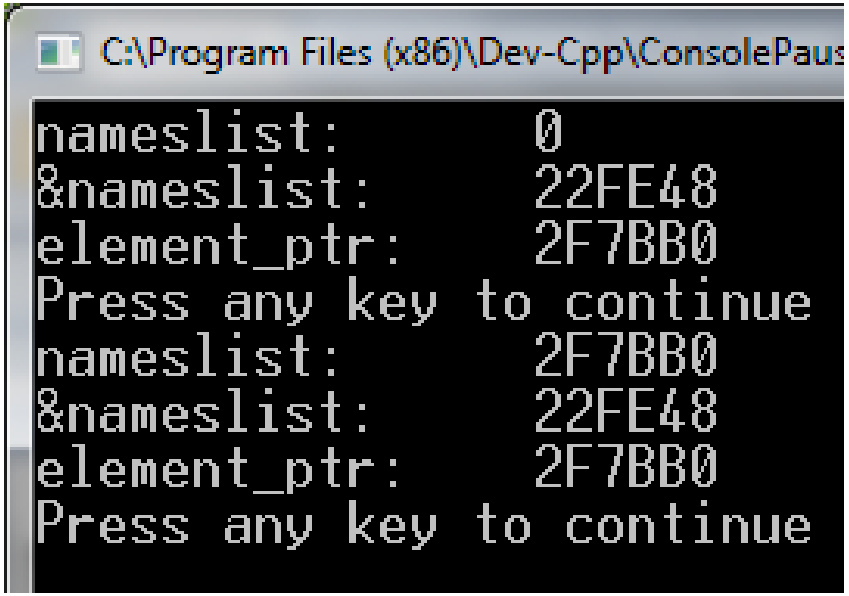
22FE48

Τιμή της namelist

2F7BB0

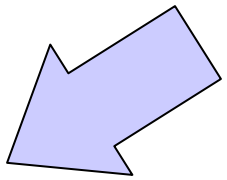
Πριν την κλήση της addtolist

Μετά την κλήση της addtolist



Διαγραφή στοιχείου

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);
}
```