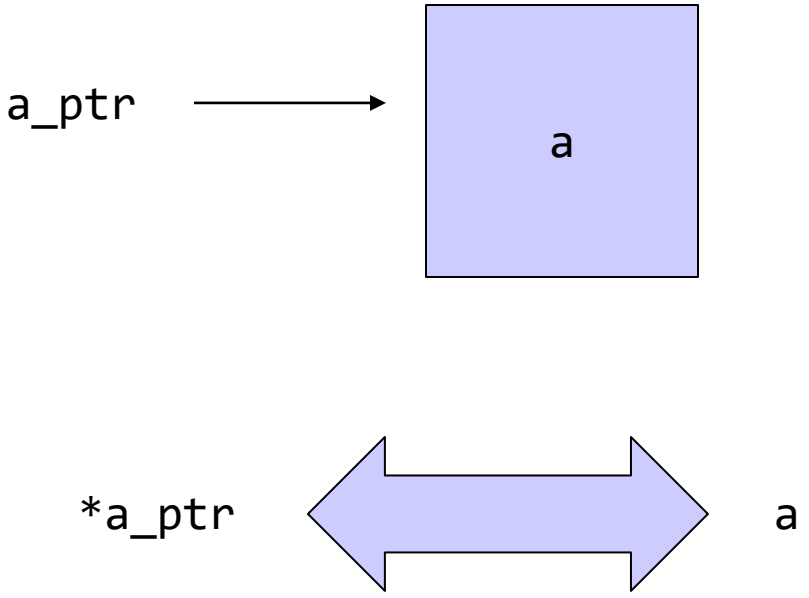
# *Διαδικαστικός Προγραμματισμός*

Βασίλης Παλιουράς
paliuras@ece.upatras.gr

# Βασικό για τα παρακάτω…

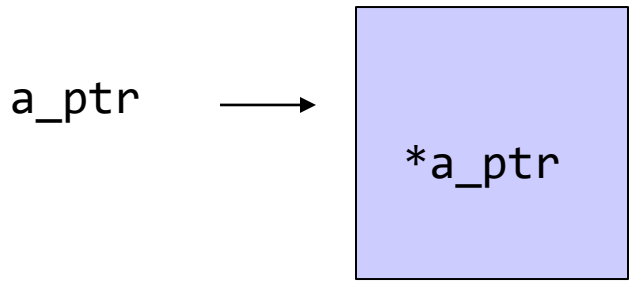
```
T * a_ptr;
T a;

a_ptr = &a;
```



# Θεμελιώδης τεχνική

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

```
T * a_ptr;

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



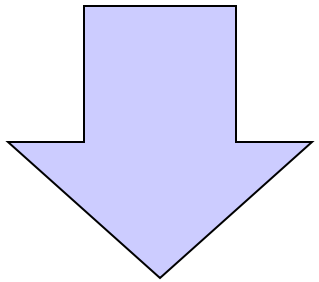

`*a_ptr` ως μεταβλητή τύπου T

`*a_ptr` ερμηνεύει τα δεδομένα στην περιοχή ως τύπου T

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

# Απλοποιημένος συμβολισμός

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



- `test_ptr->next`

```
{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;
```

# Παράδειγμα

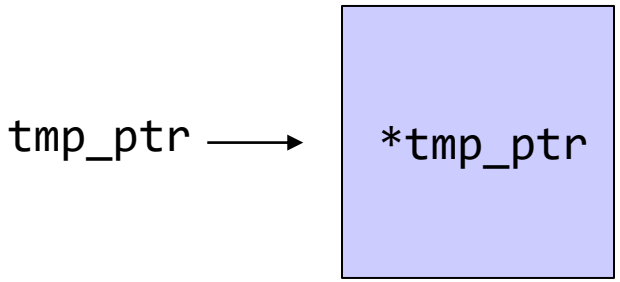
- Δήλωσε μια κενή διασυνδεμένη λίστα
- Επανάλαβε Ν φορές
  - Για έναν αριθμό, δημιούργησε έναν κόμβο
  - Τοποθέτησέ τον στη λίστα

# «Δημιούργησε κόμβο»

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

- Κατάλληλος τύπος δομής
- Δέσμευσε επαρκή περιοχή μνήμης
- Αναφερόμαστε με κατάλληλο δείκτη σε δομή στην περιοχή αυτή
- Δίνουμε τιμές στα μέλη της δομής

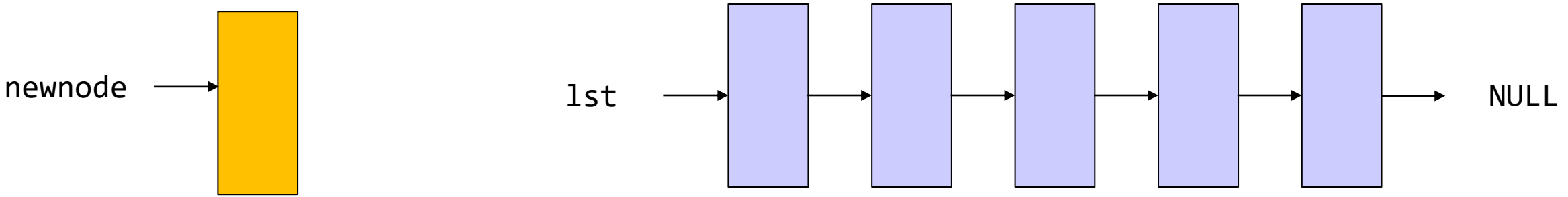
```
Node * create (int n) {
        Node * tmp_ptr;

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



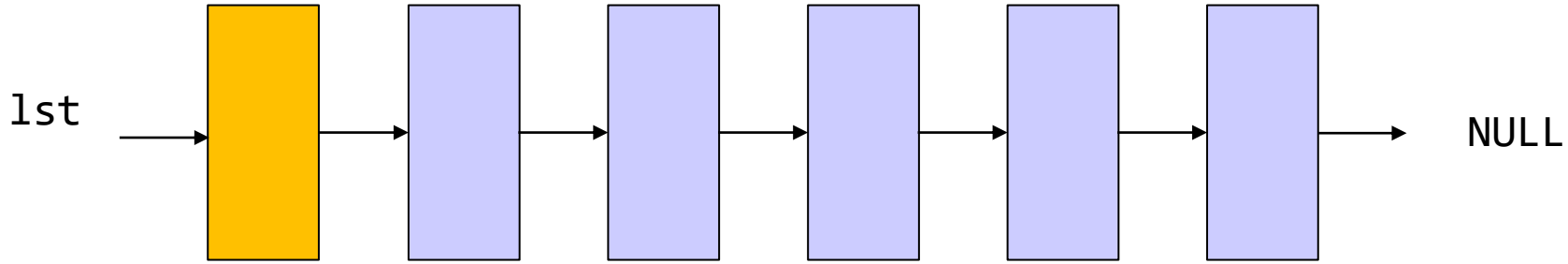
# Τοποθέτησε στη λίστα

- Στην αρχή της λίστας
- Στο τέλος της λίστας
- Σε θέση που προκύπτει από κριτήριο
  - Ως n-οστό στοιχείο
  - Με αύξουσα σειρά
  - …

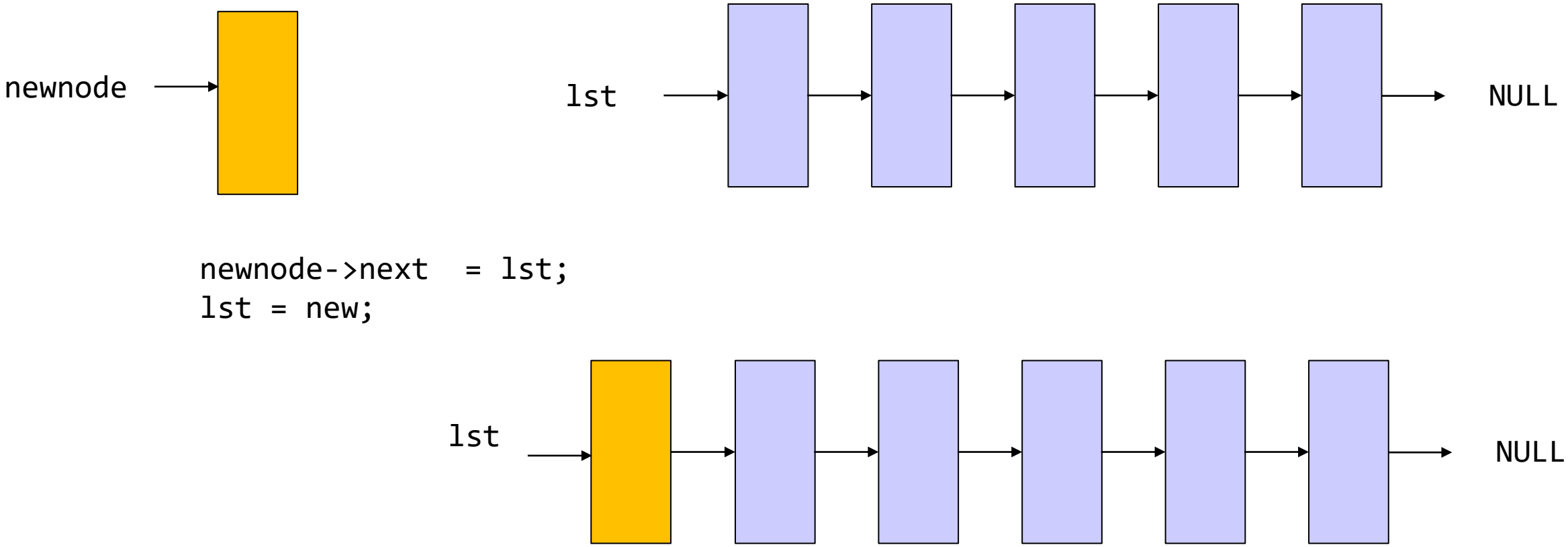
# Τοποθέτησε στην αρχή της λίστας




```
newnode->next  = lst;
lst = newnode;
```



# Τοποθέτησε στην αρχή της λίστας

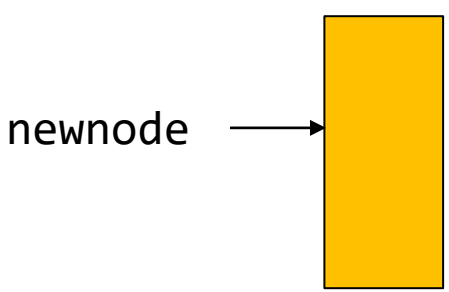


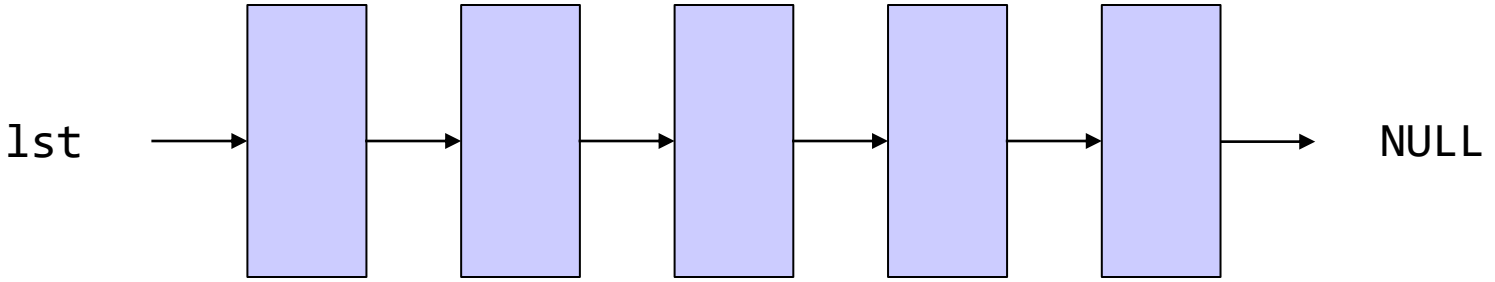

```
newnode->next  = lst;
lst = new;
```

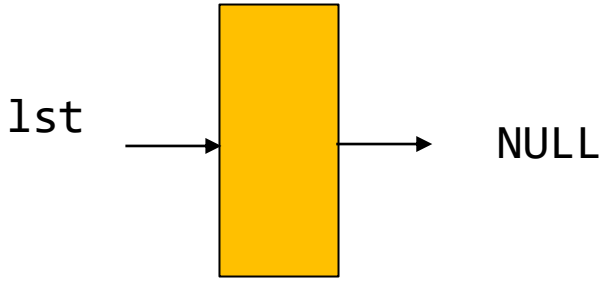
```c
#include <stdio.h>
#include <stdlib.h>
#define N 5

typedef struct d {
    int x;
    struct d * next;} Data;
typedef Data * List;

Data * createnode (int) ;
void report (List) ;

int main()
{
    List lst = NULL ;

    int i;
    for (i=0; i<N; i++) {
        Data * newnode = createnode(i); /* create a new node */
        newnode->next = lst; /* place it as first node */
        lst = newnode;     /* the list now points to the new
                            * first node */
    }
    report (lst) ;
    return 0;
}
```

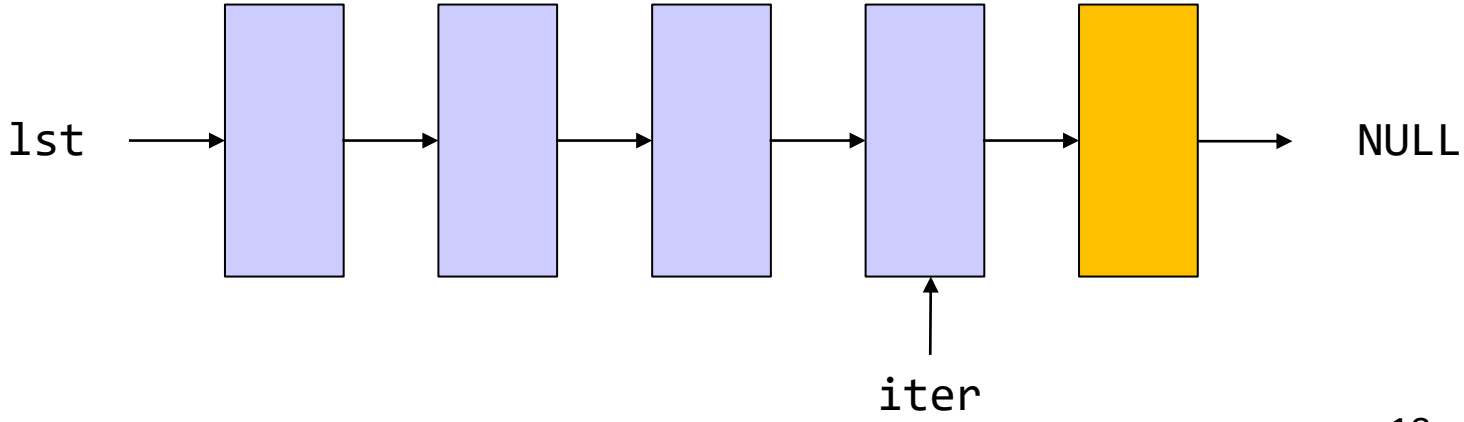
```c
Data * createnode( int d) {
    Data * temp = malloc (sizeof (Data));
    temp -> x = d;
    temp -> next = NULL;
    return temp;
}

void report(List lst) {
 Node * iter;
 for (iter = lst ; iter != NULL; iter = iter -> next){
          printf("%d:", iter->x);
  }
 return;
}
```

# Τοποθέτησε στο τέλος της λίστας



```c
#include <stdio.h>
#include <stdlib.h>
#define N 5

typedef struct d {
    int x;
    struct d * next;} Data;
typedef Data * List;

Data * createnode (int) ;
void report (List) ;
```

```c
int main()
{
    List lst = NULL ;

    int i;
    for (i=0; i<N; i++) {
        Data * newnode = createnode(i);        /* create a new node */
        if (lst==NULL) {                       /* list is empty */
            newnode->next = lst;               /* place it as only node */
            lst = newnode;     /* the list now points to the new first node */
        }
        else                /* list is not empty, find last node */
        {
            Data *iter ;
            for (iter=lst; iter->next!=NULL ; iter=iter->next) ; /* find last node */
            iter->next = newnode;                          /* append node */
        }

    }
    report (lst) ;
    return 0;
}
```

# Ειδική περίπτωση

```
int main()
{
    List lst = NULL ;
    Data * lastnode_ptr = NULL;
    int i;
    for (i=0; i<N; i++) {
        Data * newnode = createnode(i);        /* create a new node */
        if (lst==NULL) {                       /* list is empty */
            newnode->next = lst;               /* place it as first node */
            lst = newnode;            /* the list now points to the new first node */
            lastnode_ptr = newnode;
        }
        else
        {
            lastnode_ptr -> next = newnode;    /* append node to list */
            lastnode_ptr = newnode;            /* update pointer to last node */
        }

    }
    report (lst) ;
    return 0;
}
```

# Γενική περίπτωση, by value

```c
#include <stdio.h>
#include <stdlib.h>
#define N 5

typedef struct d {
    int x;
    struct d * next;} Data;
typedef Data * List;

Data * createnode (int) ;
void report (List) ;
List append (List lst, Data * newnode) ;

int main()
{
    List lst = NULL ;
    int i;

    for (i=0; i<N; i++) {
        Data * newnode = createnode(i);
        lst = append(lst, newnode);

    }
    report (lst) ;
    return 0;
}
```

```c
List append (List lst, Data * newnode) {

    if (lst==NULL) {    /* list is empty */
            newnode->next = lst; /* place it as first node */
            lst = newnode;     /* the list now points to the new first node */
            return lst;
    }
        /* list is not empty, find last element of list */
    {
            Data * iter ;
            for (iter=lst;iter->next!=NULL ; iter=iter->next) ;
            iter->next = newnode;
    }
    return lst;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#define N 5

typedef struct d {
    int x;
    struct d * next;} Data;
typedef Data * List;

Data * createnode (int) ;
void report (List) ;
void append (List * lst_ptr, Data * newnode) ;

int main()
{
    List lst = NULL ;

    int i;
    for (i=0; i<N; i++) {
        Data * newnode = createnode(i);
        append(&lst, newnode);

    }
    report (lst) ;
    return 0;
}
```

```c
void append (List * lst_ptr, Data * newnode) {

    if (*lst_ptr==NULL) {                   /* list is empty */
            newnode->next = *lst_ptr;       /* place it as first node */
            *lst_ptr = newnode;             /* the list now points to the new first node */
            return ;
    }
        /* list is not empty, find last element of list */
    {
            Data *iter ;
            for (iter=*lst_ptr;iter->next!=NULL ; iter=iter->next) ;
            iter->next = newnode;
    }
    return ;
}
```

# Παράδειγμα

- Παραγωγή ψευδοτυχαίων αριθμών και καταχώρησή τους σε απλά διασυνδεδεμένη λίστα
- Στη διασυνδεμένη λίστα κάθε αριθμός καταχωρείται μόνο μία φορά. Καταγράφουμε
  - ποιοι αριθμοί εμφανίστηκαν και
  - πόσες φορές ο καθένας.

Δήλωσε μια κενή διασυνδεδεμένη λίστα

Επανάλαβε Ν φορές:

Δημιούργησε έναν τυχαίο αριθμό

Αν η λίστα είναι κενή, φτιάξε έναν κόμβο με τον αριθμό στη λίστα και πρόσθεσέ τον στη λίστα

Αλλιώς (όχι κενή)

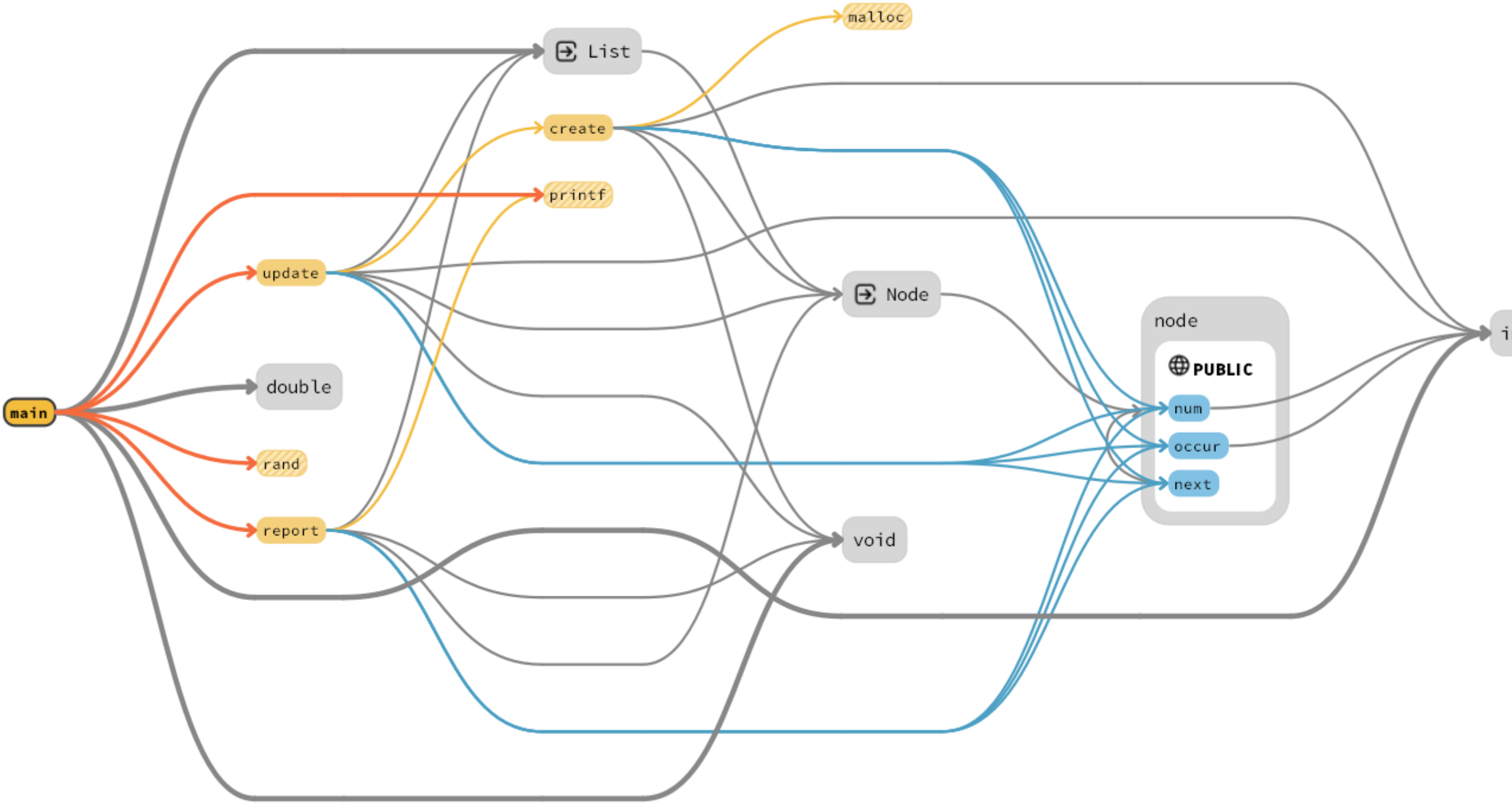
Αναζήτησε τον αριθμό στη διασυνδεμένη λίστα

Αν υπάρχει ήδη, αύξησε τις εμφανίσεις κατά μία

Αν δεν υπάρχει, πρόσθεσε νέο κόμβο στη λίστα

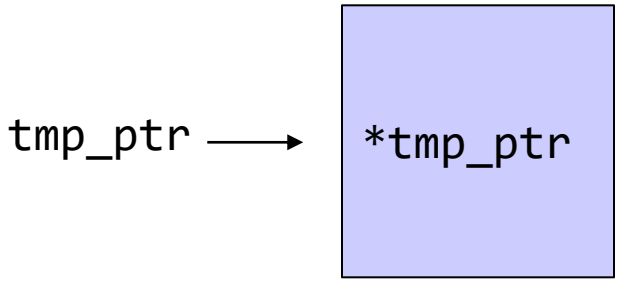
- Αν δεν υπάρχει στη λίστα, πρόσθεσε νέο κόμβο στη λίστα

- Διατρέχω τη διασυνδεμένη λίστα και αν δεν βρω το στοιχείο, συμπεραίνω ότι δεν υπάρχει.

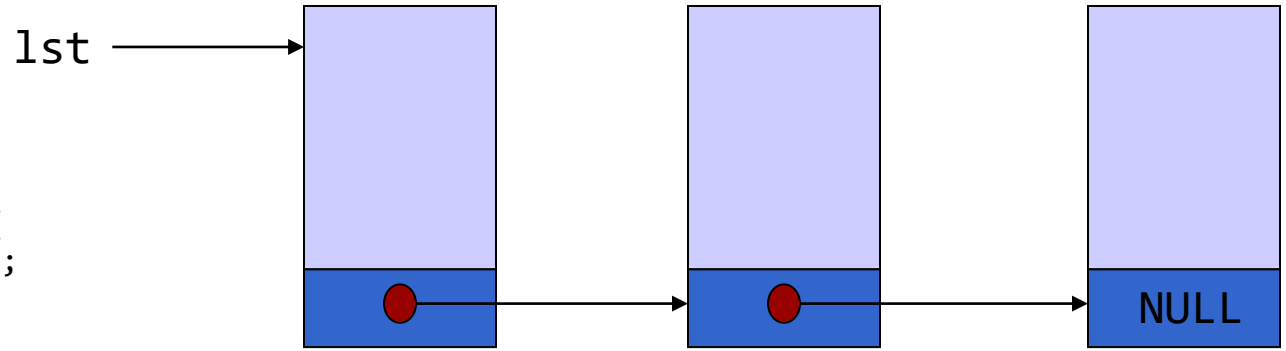

main
double
rand
report
update
List
create
printf
malloc
Node
void
node
PUBLIC
num
occur
next

```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);       /* by value call and returns possibly new value */

                printf("after: %p\n", (void *) mylst);
        }

        report(mylst);

        return EXIT_SUCCESS ;
 }
```

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

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



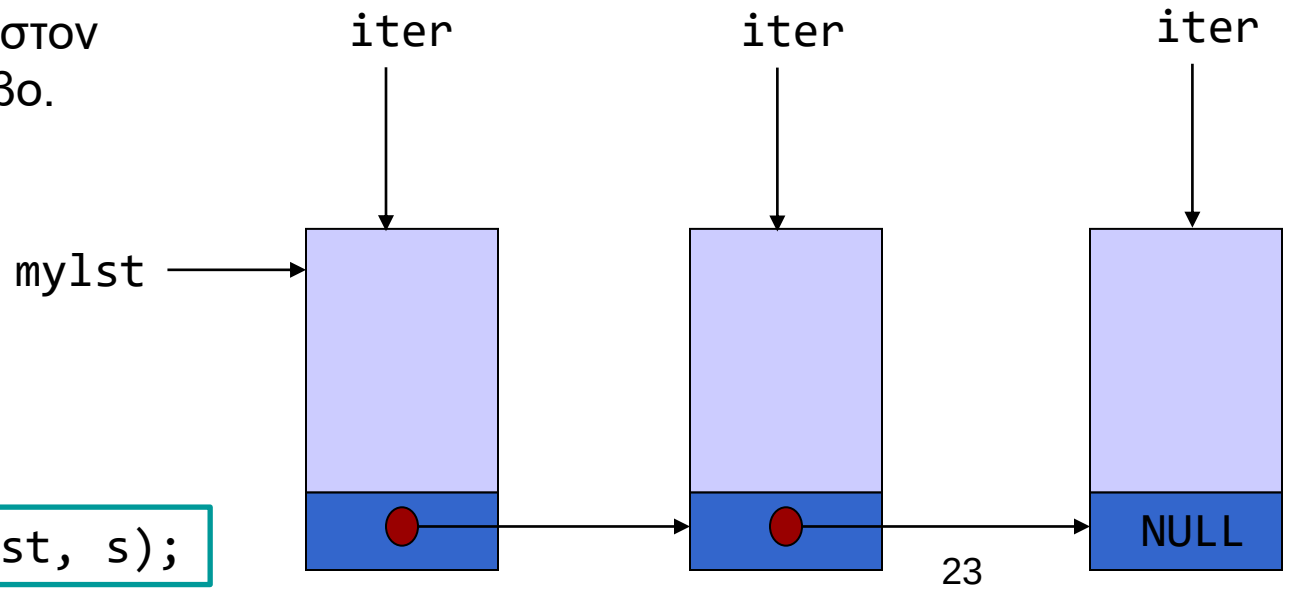

```
void report(List lst) {
        Node * iter;

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



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

        if (mylst == NULL) {  /* list is empty */
                temp_list = create(n);
                return temp_list;
        }

        for (iter = temp_list; iter->next != NULL; iter = iter -> next) {
                if (iter -> num == n) {
                        (iter -> occur)++;    /* number exists */
                        return temp_list;
                        }
                }

        if (iter->num==n) {
                (iter->occur)++;
                return temp_list;
                }

        iter->next = create(n);

        return temp_list;
}
```

Δήλωσε μια κενή διασυνδεδεμένη λίστα
Επανάλαβε Ν φορές:
    Δημιούργησε έναν τυχαίο αριθμό
        Αν η λίστα είναι κενή, φτιάξε έναν κόμβο με τον αριθμό στη λίστα
                                και πρόσθεσέ τον στη λίστα
        Αλλιώς (όχι κενή)
            Αναζήτησε τον αριθμό στη διασυνδεμένη λίστα
                Αν υπάρχει ήδη, αύξησε τις εμφανίσεις κατά μία
                Αν δεν υπάρχει, πρόσθεσε νέο κόμβο στη λίστα

Όταν τερματίζεται το for,
Το iter δείχνει στον τελευταίο κόμβο.




```
mylst = update(mylst, s);
```

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

          if (mylst == NULL) {                        /* empty list */
                    temp_list = create(n);
                    return temp_list;
          }
          for (iter = temp_list; iter!= NULL; iter = iter -> next) {
                    if (iter -> num == n) {
                              (iter -> occur)++;
                              return temp_list;     /* if found, count and return */
                    }

                   if (iter->next == NULL) {         /* if last node  and not found
                      iter->next = create(n);         * create new node and append to list
                      return temp_list;               */
                   }
          }

    return temp_list;
}
```

```
mylst = update(mylst, s);
```

```
void updateref (List *mylst_ptr, int n) {
        List temp_list = *mylst_ptr;
        Node * iter;

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

        return ;
}
```

```
updateref(&mylst, s);
```

# Τύποι λίστας στο mytypes.h

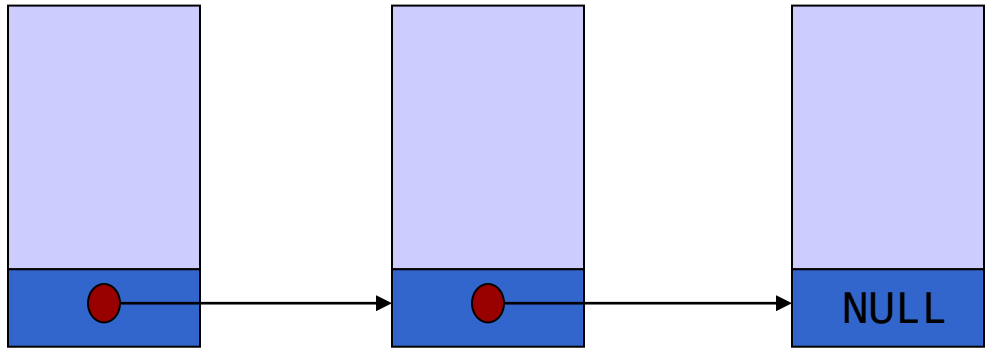
- Listelement
  - Στοιχείο λίστας (**struct** listelement)
  - **typedef struct** listelement Listelement ;
- Listelement_ptr
  - Δείκτης σε στοιχείο λίστας. Ίδιο με
    - **struct** listelement *
    - Listelement *
    - **typedef** Listelement * Listelement_ptr;
- List
  - Δείκτης σε στοιχείο λίστας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

# Δημιουργία στοιχείου

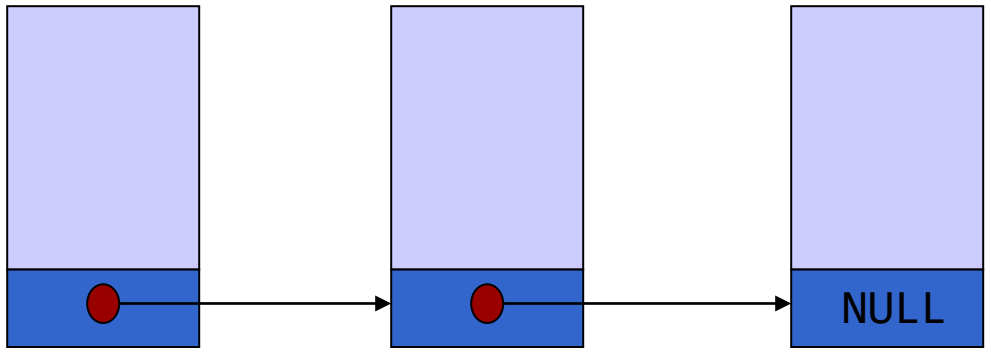
```
#include <string.h>
#include <stdlib.h>
#include "mytypes.h"

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

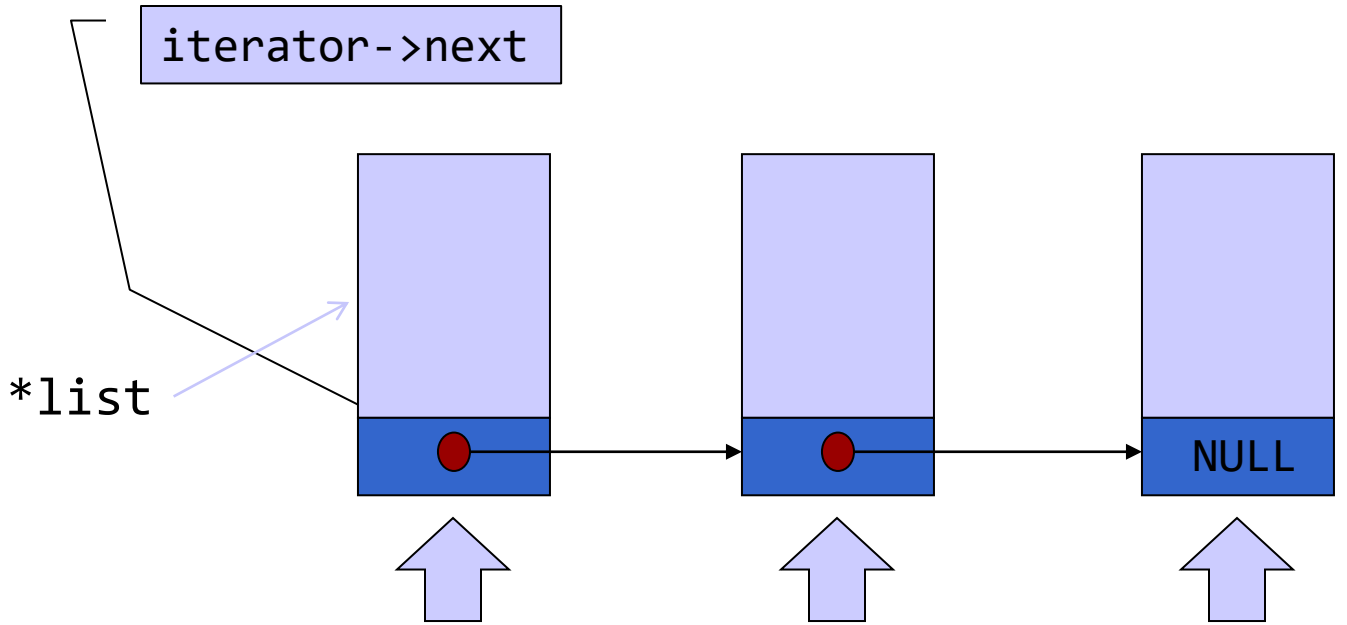



Παράμετρος δείκτης σε λίστα ή διπλός δείκτης σε στοιχείο

```
#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.

```
iterator = *list                    iterator->next=NULL
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;
};
```

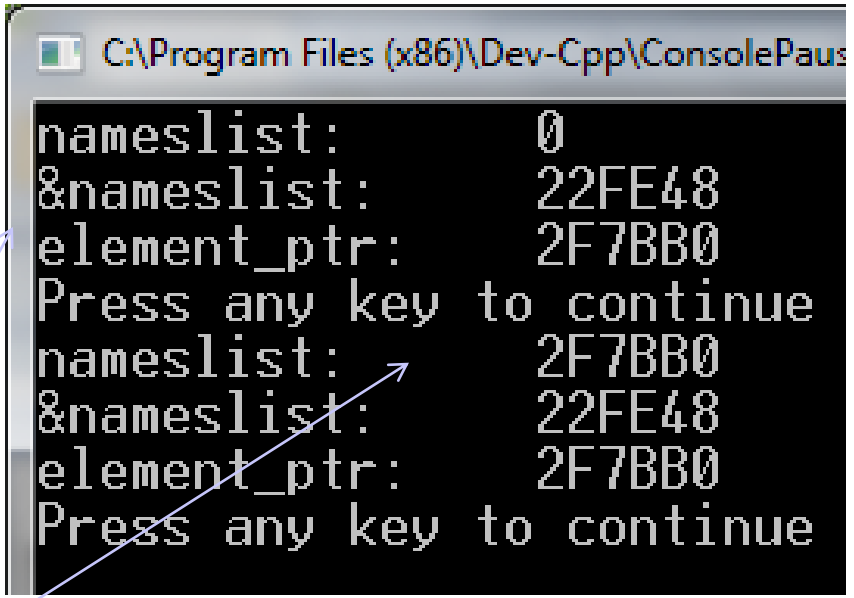
# Λίστα ως παράμετρος με αναφορά

```
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)

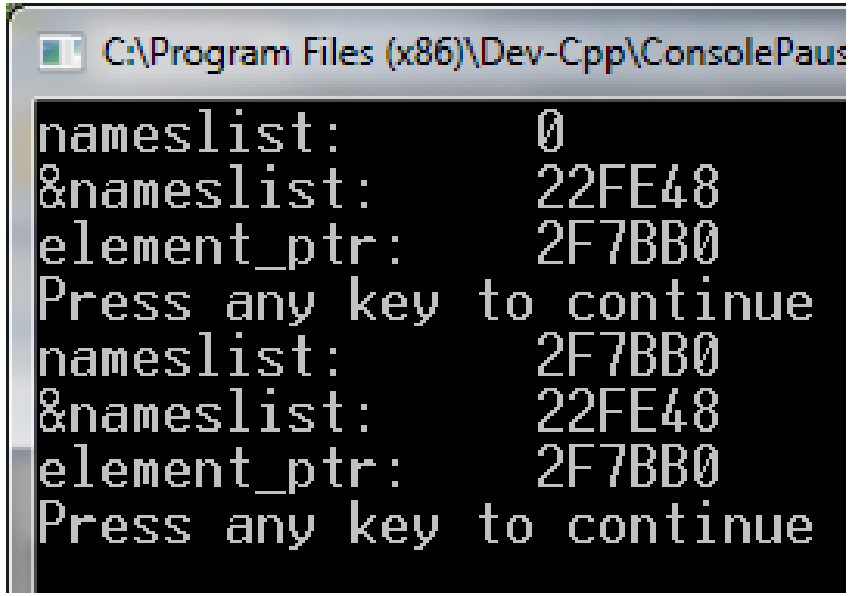
22FE48

Τιμή της namelist

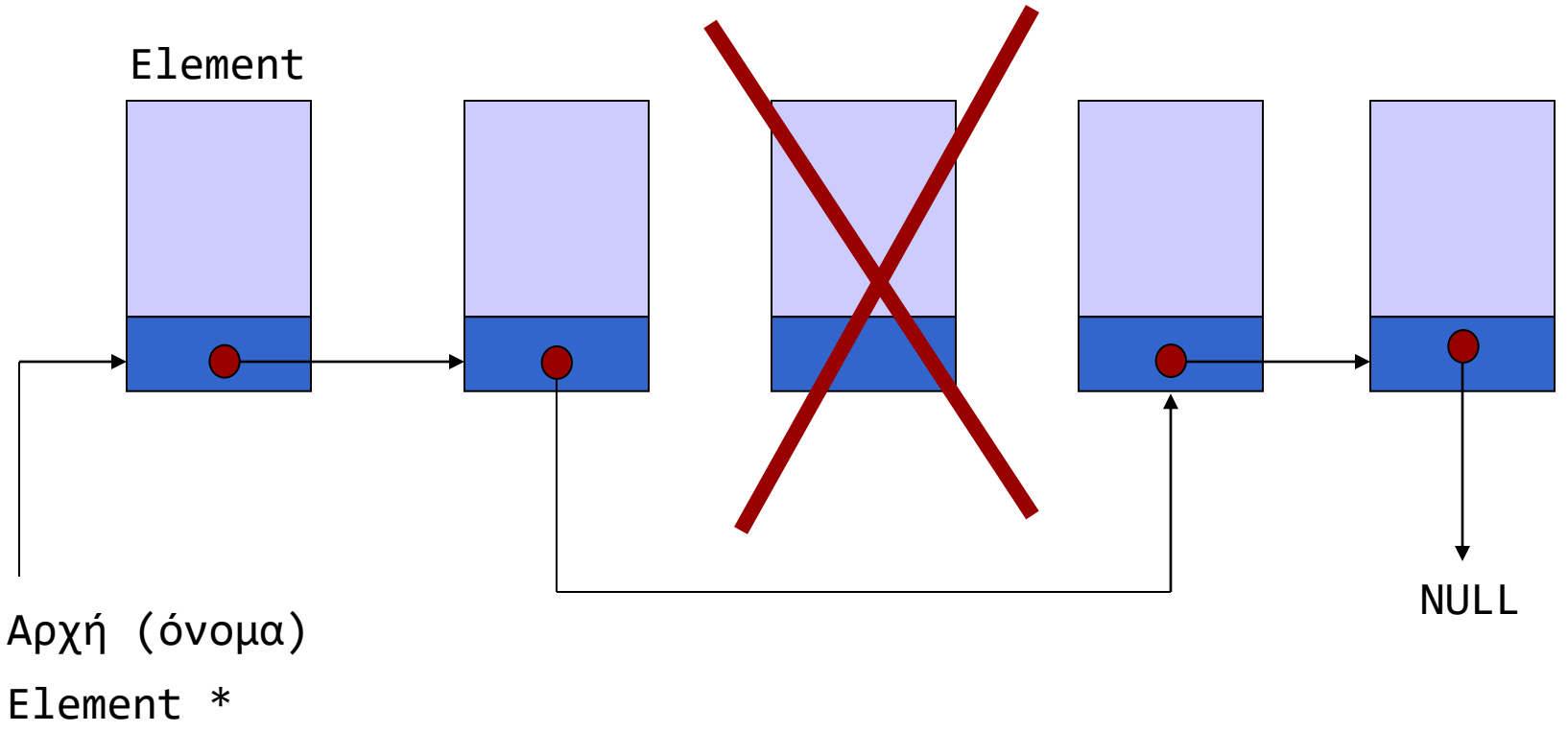
2F7BB0

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 != 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`

# Εισαγωγή στοιχείου στη θέση pos (1/3)

```c
#include <stdio.h>
#include <stdlib.h>
#define N 5

typedef struct d {
    int x;
    struct d * next;} Data;
typedef Data * List;

Data * createnode (int) ;
void report (List) ;
void append (List * lst_ptr, Data * newnode) ;
List insert(List lst, int pos, Data * node_ptr);

int main()
{
    List lst = NULL ;

    int i;
    for (i=0; i<N; i++) {
        Data * newnode = createnode(i); /* create a new node */
        append(&lst, newnode);
        report (lst) ;
    }

    lst = insert(lst, 0, createnode(100)); /* insert at pos 0 */
    report (lst) ;
    lst = insert(lst, 1, createnode(400)); /* insert at pos 1 */
    report (lst) ;
    lst = insert(lst, 200, createnode(400)); /* (try to) insert at pos 200 */

    report (lst) ;

    return 0;
}
```

# Εισαγωγή στοιχείου στη θέση pos (2/3)

```
List insert(List lst, int pos, Data * node_ptr) {
    List local_list = lst;
    Data * iter;
    int i ;

    if (node_ptr == NULL) {
                printf("no node to be inserted");
                return local_list;
    }

    if (pos==0) {   /                          * insert if first node */
        node_ptr->next = local_list;
        return node_ptr;
    }

    for (i=0, iter = lst; i < pos-1 && iter!=NULL; i++, iter=iter->next) ;

    if (iter == NULL) {
          printf("too few nodes in list, not inserted\n");
          return local_list;
          }

    /* iter points at (pos-1)st node */
    node_ptr->next = iter->next;
    iter->next = node_ptr;

    return local_list;
}
```

# Υποστηρικτικές συναρτήσεις (3/3)

```
void append (List * lst_ptr, Data * newnode) {

    if (*lst_ptr==NULL) {                    /* list is empty */
            newnode->next = *lst_ptr;        /* place it as first node */
            *lst_ptr = newnode;              /* the list now points to the new first node */
            return ;
    }
        /* list is not empty, find last element of list */
    {
            Data *iter ;
            for (iter=*lst_ptr;iter->next!=NULL ; iter=iter->next) ;
            iter->next = newnode;
    }
    return ;
}

Data * createnode( int d) {
    Data * temp = malloc (sizeof (Data));
    temp -> x = d;
    temp -> next = NULL;
    return temp;
}

void report (List lst) {
    if (lst == NULL) { printf("\n") ; return ;}
    printf("%d:", lst->x);
    report(lst->next);
    return;
}
```

# Εισαγωγή σε θέση pos, by reference

```c
#include <stdio.h>
#include <stdlib.h>
#define N 5

typedef struct d {
    int x;
    struct d * next;} Data;
typedef Data * List;

Data * createnode (int) ;
void report (List) ;
void append (List * lst_ptr, Data * newnode) ;
List insert(List lst, int pos, Data * node_ptr);
int insertref(List * lst, int pos, Data * node_ptr) ;
int main()
{
    List lst = NULL ;
    int status ;
    int i;
    for (i=0; i<N; i++) {
        Data * newnode = createnode(i); /* create a new node */
        append(&lst, newnode);
        report (lst) ;
    }

    printf("%d\n", status = insertref(&lst, 0, createnode(100)));
    /* could exploit status to check if attempt succeeded */
    report (lst) ;
    printf("%d\n", status = insertref(&lst, 3, createnode(400)));
    report (lst) ;
    printf("%d\n", status = insertref(&lst, 200, createnode(500)));
    report (lst) ;


    return 0;
}
```

Η λίστα περνάει ως παράμετρος by reference,
Επιστρέφεται με return κωδικός τύπου int που
Δηλώνει αν η διαδικασία ολοκληρώθηκε επιτυχώς

```
int insertref(List * lst_ptr, int pos, Data * node_ptr) {
/*   returns 0 upon successful completion
 *           1 for empty node to be inserted
 *           2 for two few nodes
 */

    Data * iter;
    int i ;

    if (node_ptr == NULL) {printf("no node to be inserted"); return 1;}

    if (pos==0) {   /* insert first node */
        node_ptr->next = *lst_ptr;
        *lst_ptr = node_ptr;
        return 0;
    }

    for (i=0, iter = *lst_ptr; i < pos-1 && iter!=NULL; i++, iter=iter->next) ;
    if (iter == NULL) {printf("too few nodes in list, not inserted\n"); return 2;}

    /* iter points at (pos-1)st node */
    node_ptr->next = iter->next;
    iter->next = node_ptr;

    return 0;
}
```

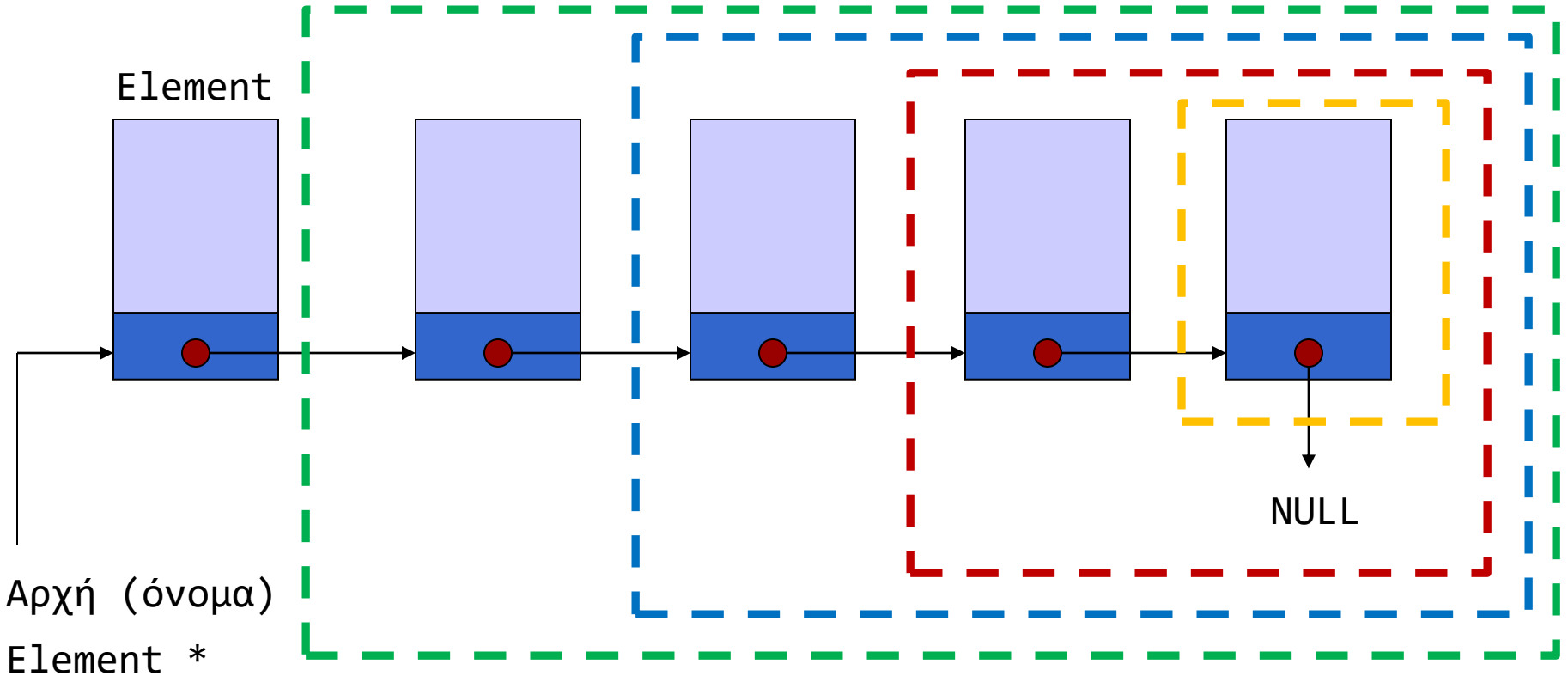
# Χρήσιμα σημεία σε συναρτήσεις επεξεργασίας λίστας

- 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);
}
```

# Αυτοαναφορική (self-referential) δομή




Ένας κόμβος δείχνει σε κόμβο ίδιου τύπου (ή NULL)

Άλλη ανάγνωση: Κάθε κόμβος δείχνει σε μικρότερη λίστα (ή NULL)

# 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);
}
```

# Απλοποίηση κώδικα

- Αξιοποίηση αυτοαναφορικής δομής
- Αναδρομική διαχείριση λίστας

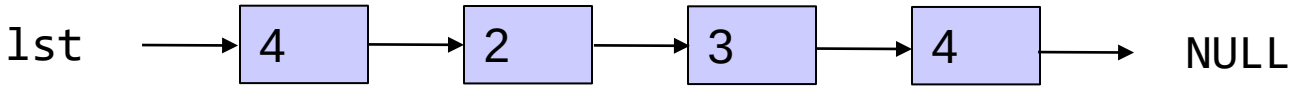
```
struct node {
        int value ;
        struct node * next;
};
typedef struct node Node;
typedef struct node * Node_ptr;
typedef struct node * List;
```

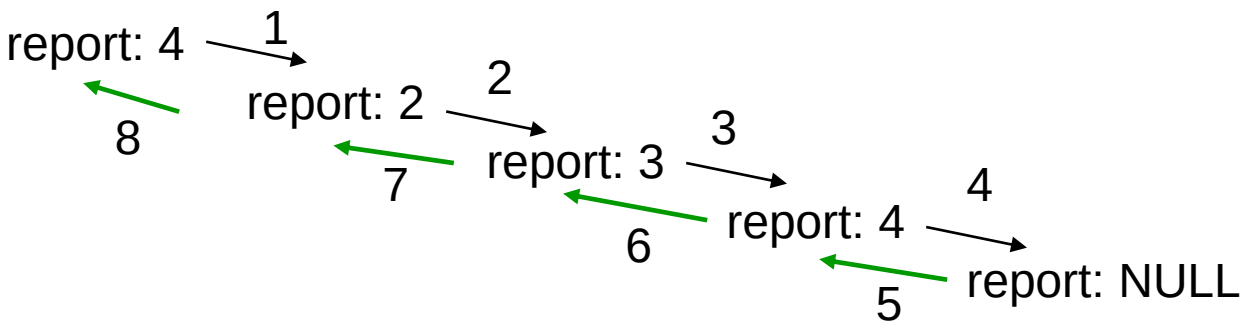
```
Node_ptr createnode(int a);
void report(List lst) ;
void append ( List * list_ptr, Node_ptr node_ref);
```

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

int main(void ) {

        int i ;
        int temp ;
        List mylst = NULL;
        Node_ptr node_ref;

        srand(0);

        for (i=0; i<N; i++) {
                temp = rand() % 10 ;
                node_ref = createnode(temp);
                append( &mylst, node_ref) ;
        }

        report(mylst);

        return EXIT_SUCCESS;
}
```

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

Node_ptr createnode(int a) {
        Node_ptr new_node_ptr;

        new_node_ptr = malloc( sizeof (Node) ) ;
        new_node_ptr -> value = a;
        new_node_ptr -> next = NULL;

        return new_node_ptr;

}
```

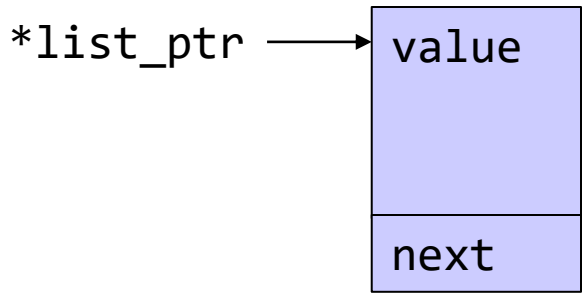
```
void report(List lst) {

        if (lst == NULL) return;

        printf("%d ", lst -> value);
        report ( lst->next);

        return ;
}
```




Να θυμηθούμε το stack, recursion depth limits

```
void append ( List * list_ptr, Node_ptr node_ref) {

        if ( *list_ptr == NULL) {
                *list_ptr = node_ref;
                return ;
        };

        append( &( (*list_ptr)->next ), node_ref );

        return ;
}
```




(*list_ptr)->next είναι lvalue

Βρίσκεται στη διεύθυνση &((*list_ptr)->next)