ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν
Πανεπιστήμιον Αθηνών

Δομές Δεδομένων και Τεχνικές Προγραμματισμού

Ενότητα 4: ΑΤΔ Λίστα

Ιωάννης Κοτρώνης
Σχολή Θετικών Επιστημών
Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Σκοποί ενότητας

- Ορίζει τον ΑΤΔ Λίστα
- Να αναδείξει το πρόβλημα της μετακίνησης στοιχείων με τον σχεδιασμό με πίνακα
- Να εισάγει την σχεδιαστική επιλογή με δυναμικές συνδεδεμένες δομές με struct+pointers.
- Να αναδείξει εναλλακτικούς σχεδιασμούς και αντίστοιχες υλοποιήσεις
- Να εισάγει την σχεδιαστική επιλογή με στατικές συνδεδεμένες δομές με πίνακα.

Περιεχόμενα ενότητας

- Ορισμός και χρήσεις ΑΤΔ Λίστας
- Σχεδιασμός και υλοποίηση με πίνακα
- Σχεδιαστική Επιλογή με δυναμικές συνδεδεμένες δομές με struct+pointers
- Εναλλακτικοί σχεδιασμοί (με κεφαλή, διπλά συνδεδεμένη, κυκλική, με ένδειξη τέλους, ταξινομημένη) και υλοποιήσεις
- Σχεδιαστική Επιλογή συνδεδεμένης δομής με πίνακα.

Ενότητα 4

ΑΤΔ Λίστα

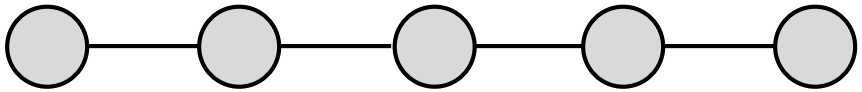
ΛΙΣΤΕΣ

- Ορισμός ΑΤΔ Λίστα
- ΑΤΔ Ακολουθιακή Λίστα
- Διαχείριση Δεικτών και Λιστών στη C
- ΑΤΔ Συνδεδεμένη Λίστα
  - Υλοποίηση με δείκτες (pointers)
  - Υλοποίηση με πίνακα
- Εφαρμογές και Χρήση Λιστών

Λίστες (Lists)

Δεδομένα

Ο ΑΤΔ λίστα είναι μια πεπερασμένη συλλογή στοιχείων του ίδιου τύπου Τ με γραμμική δομή (ολική διάταξη).



Βασικές Πράξεις

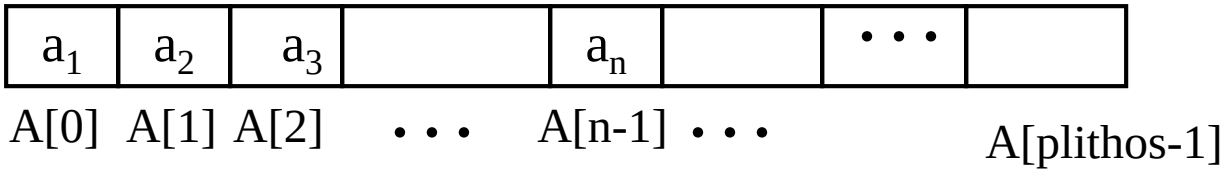
- Δημιουργία
- Καταστροφή
- Κενή
- Εισαγωγή
- Διαγραφή
- Περιεχόμενο (τιμή στοιχείου)
- Επόμενο, προηγούμενο στοιχείο
- Αναζήτηση

Σχεδιασμός Λίστας με πίνακα

Στη συνέχεια θα χρησιμοποιήσουμε πάλι τον πίνακα σαν το μέσο αποθήκευσης των στοιχείων μιας λίστας όπως ακριβώς έγινε για τη στοίβα και την ουρά.

Αυτή η **ακολουθιακή υλοποίηση** έχει σαν βασικό χαρακτηριστικό ότι διαδοχικά στοιχεία της λίστας αποθηκεύονται σε διαδοχικές θέσεις του πίνακα. Έτσι το πρώτο στοιχείο της λίστας αποθηκεύεται στην πρώτη θέση του πίνακα, το δεύτερο στοιχείο της λίστας αποθηκεύεται στη δεύτερη θέση του πίνακα κ.ο.κ.

Σχεδιασμός



Η λογική διάταξη των στοιχείων της λίστας αντιστοιχεί τη φυσική διάταξη των στοιχείων του πίνακα.

Εκτός από την αποθήκευση των στοιχείων της λίστας στον πίνακα είναι αναγκαίο να χρησιμοποιηθεί και μια βοηθητική μεταβλητή megethos που θα διατηρεί το εκάστοτε πλήθος των στοιχείων της λίστας.
Οδηγούμαστε λοιπόν στις δηλώσεις :

```
#define plithos ...
typedef ... typos_stoixeiou;

typedef struct {
     typos_stoixeiou typos_pinaka[plithos]
     int megethos;
} typos_listas;

typos_listas lista;
```

Η υλοποίηση των βασικών πράξεων για τον ΑΤΔ ακολουθιακή λίστα είναι εύκολη και ορισμένες από αυτές παρουσιάζονται παρακάτω:

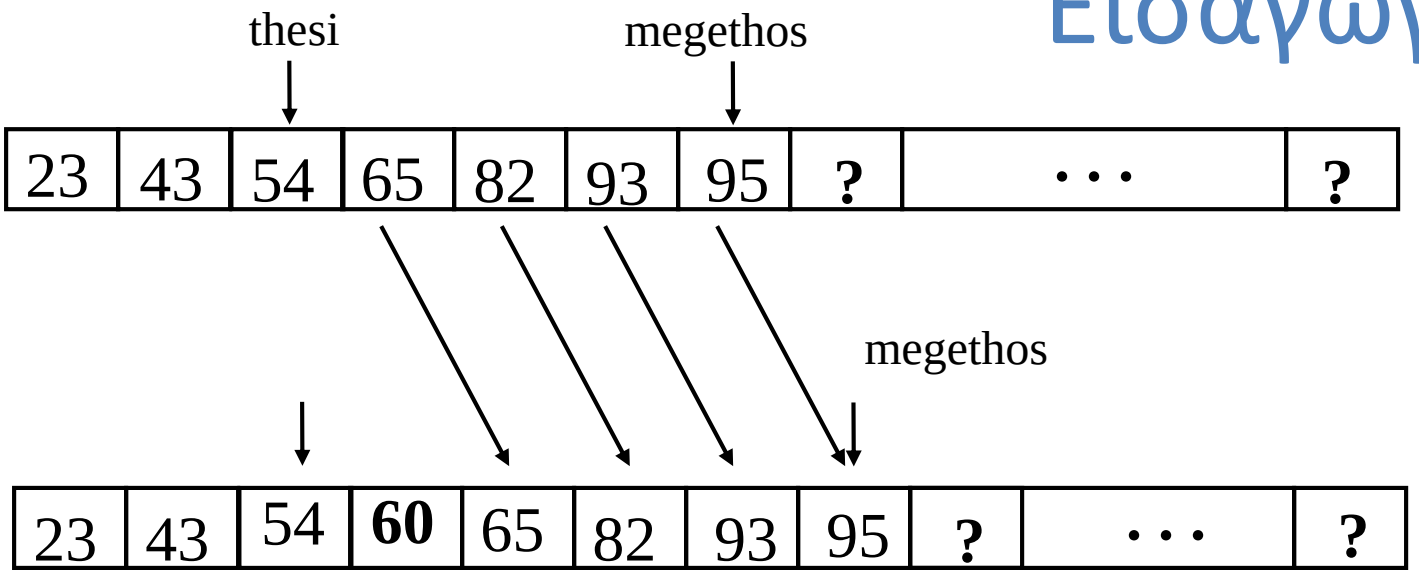
```
void dimiourgia (typos_listas *lista)
/* Προ : Καμμία
   Μετα : Δημιουργεί μια κενή λίστα/*
{
     lista->megethos = 0;
}
```

```
int keni(typos_listas  lista)
/* Προ : Δημιουργία μιας λίστας.
   Μετα : Ελέγχει αν μια λίστα είναι κενή*/

{
     return (lista.megethos == 0 );
}
```

```
int epomenos(typos_listas lista, int thesi){
/* Προ : Δημιουργία λίστας.
   Μετά: Αν είναι έγκυρη η Thesi επιστρέφει την επόμενη
      θέση της thesi αλλιώς -1.
*/

      if ((thesi<0)||(thesi>lista.megethos-1)){
            return  (-1);
      }else
            return (++thesi);
}
```

Παρατηρήστε την διπλή χρήση της τιμής επιστροφής της συνάρτησης. Εφόσον δεν έχουν νόημα οι αρνητικές τιμές σε θέση πίνακα, χρησιμοποιούμε την αρνητική τιμή -1 για ένδειξη λάθους. Εναλλακτικός τρόπος αντί για σημαία.

Εισαγωγή

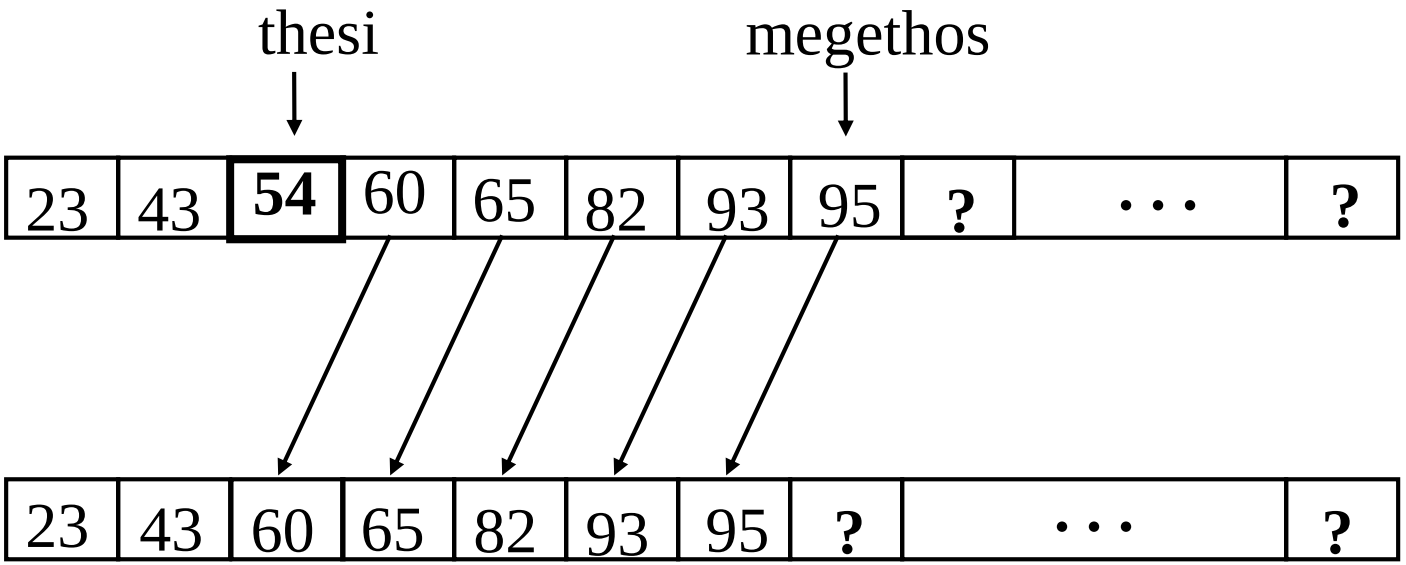


Μόλις δημιουργηθεί ο χώρος αυτός τότε τοποθετείται το νέο στοιχείο της λίστας. Η υλοποίηση της διαδικασίας της εισαγωγής ενός νέου στοιχείου στη λίστα παρουσιάζεται στη συνέχεια. Όπως για την ουρά και τη στοίβα, έτσι και για τη λίστα θα πρέπει να ελεγχθεί πρώτα αν η λίστα είναι γεμάτη.

Πολυπλοκότητα : O(n)

```
int eisagogi_meta(typos_listas *lista,
                  typos_stoixeiou stoixeio, int thesi, int *full){
/* Προ : Δημιουργία της λίστας.
   Μετά: Εισάγεται το στοιχείο stoixeio μετά την θέση thesi
*/
        int i;
        *full = 0;
        if (thesi<0)||(thesi>lista->megethos-1)
                return(-1);
        else if (lista->megethos == plithos)
             *full =1;
        else {/* εισαγωγή είναι δυνατή. Μετακίνηση στοιχείων μια
                        θέση δεξιά από την thesi*/
                for (i = lista->megethos-1 ; i >= thesi+1 ; i--)
                lista->pinakas[i+1] = lista->pinakas[i];

                lista->pinakas[thesi+1]=stoixeio;
                lista->megethos++;
        }
        return(0);
}
```

Διαγραφή Στοιχείου 54 (Μετακινήσεις)



```
int diagrafi(typos_listas *lista, int thesi)
/* Προ : Δημιουργία της λίστας.
   Μετά: Διαγράφεται το στοιχείο της *lista στη θέση thesi.
*/
{       int i;
        if ((thesi<0)||(thesi>lista->megethos -1))
                return(-1)
        else
        if (keni(*lista))
                *empty=1;
        else  {/* μείωση του μεγέθους της λίστας κατά 1
                        και αφαίρεση του κενού χώρου */
                lista->megethos--;
                for (i=thesi; i <= lista->megethos-1 ; i++ )
                        lista->pinakas[i] = lista->pinakas[i+1];
        }
        return(0);
}
```

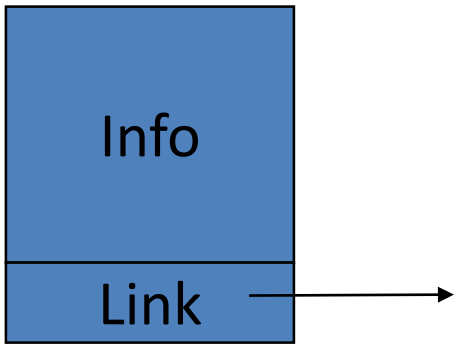
Σχεδιαστική Επιλογή με Δυναμικές Δομές

Δείκτες (Pointers) και Structs

Ένας βασικός «δομικός λίθος». Όλη μαζί η δομή (struct) ονομάζεται κόμβος (Node) και αποτελείται από

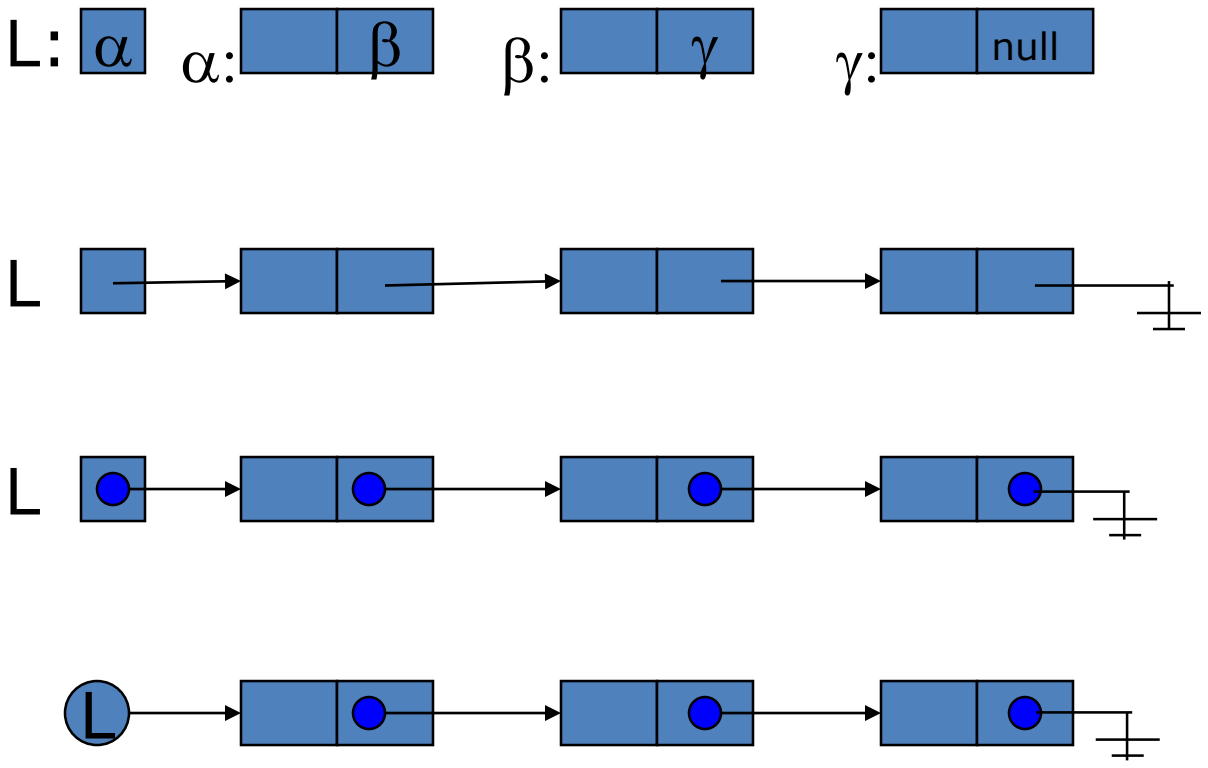
- Το μέλος **Info** οποιοδήποτε τύπου, έστω InfoField.

- Το μέλος **Link** τύπου δείκτη σε Node.

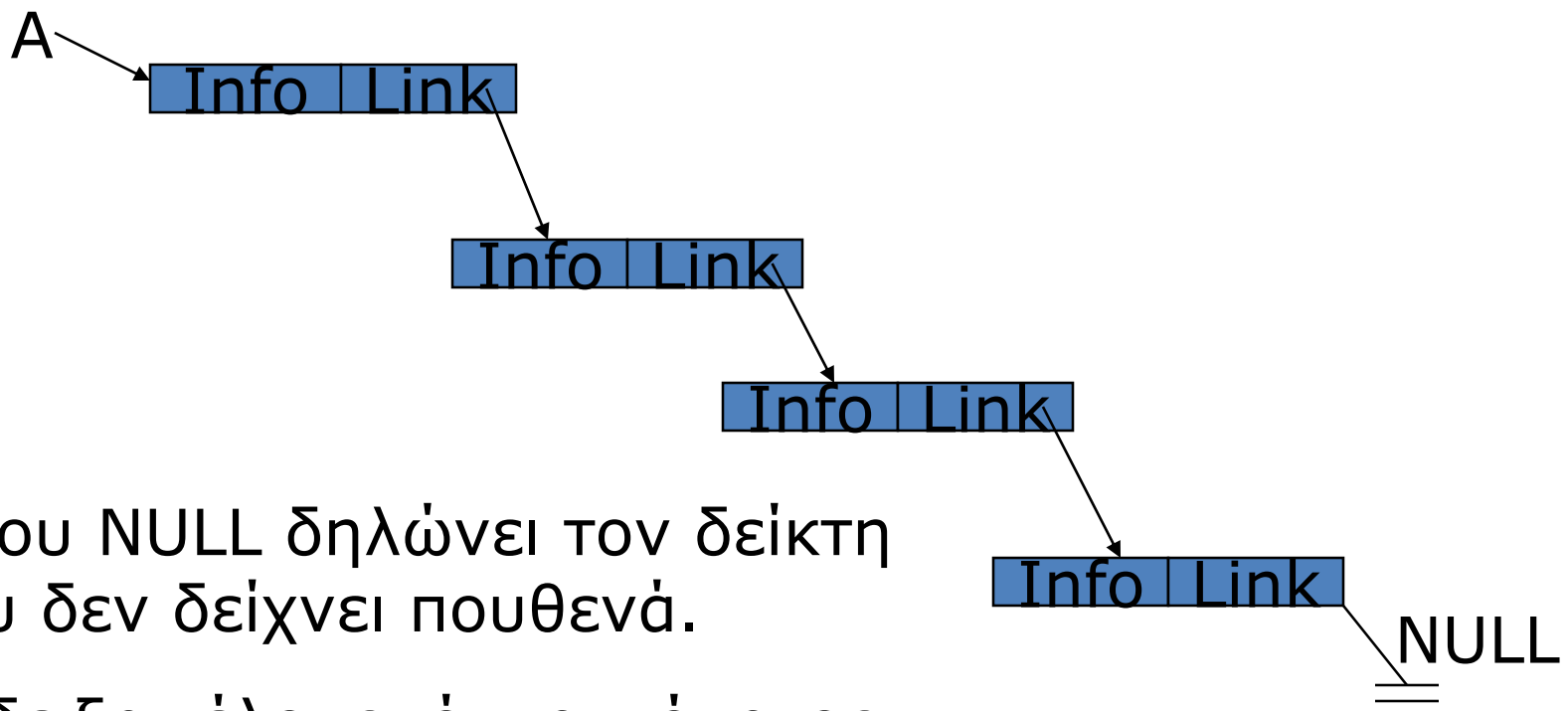



```
typedef struct NodeTag {
        InfoField Info;
        struct NodeTag *Link;
} NodeType;
```

Απεικονίσεις Λιστών και ενδείξεις τέλους τους



Χρησιμοποιούμε εικόνες του τύπου



Όπου NULL δηλώνει τον δείκτη που δεν δείχνει πουθενά.

Ένδειξη τέλους, όχι αυτόματος τρόπος. (*NULL).x είναι σφάλμα (όχι πάντα διαγνώσιμο)

Προσοχή σε …

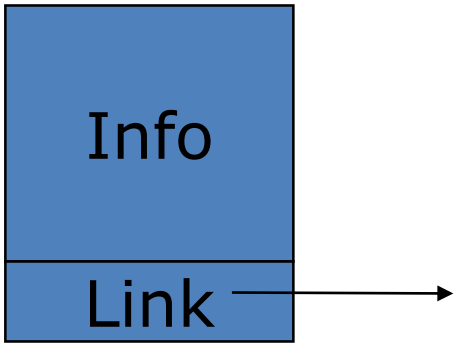
Lifetime-Διάρκεια Ζωής (3 κατηγορίες μνήμης):

- Καθολικές στατικές μεταβλητές
- Auto-Stack (τοπικές). Αυτόματη διαχείριση (μπλοκ).
- Heap. Οποιοδήποτε αντικείμενο δημιουργείται με **malloc** ισχύει μέχρι να το αποδεσμεύσουμε με **free**. Η διαχείριση Heap είναι ευθύνη του προγραμματιστή. **Ανακύκλωση:** Όταν δεν χρειαζόμαστε άλλο τότε **free**.

Alias-Συνώνυμα: Πολλαπλοί τρόποι αναφοράς για το ίδιο αντικείμενο στη μνήμη.

Αιωρούμενοι (dangling) δείκτες: τι συμβαίνει αν ένας δείκτης δείχνει σε θέση μνήμης που δεν ισχύει;

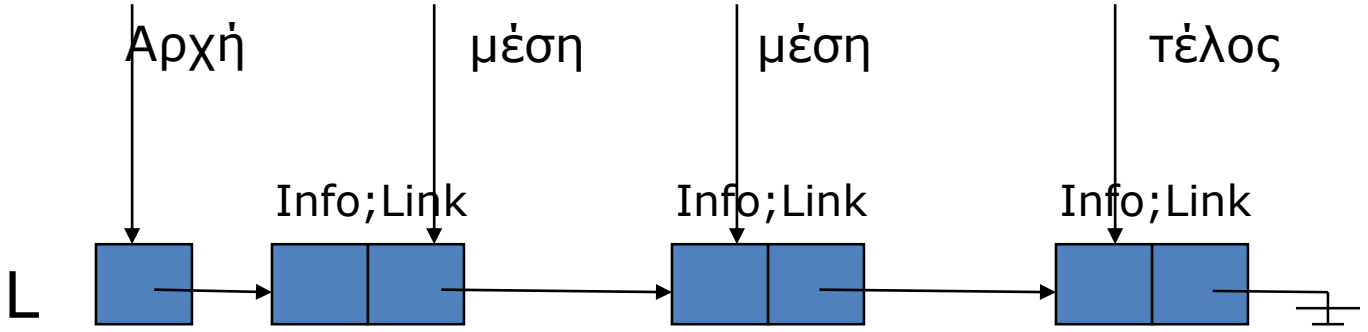
Συνδεδεμένη λίστα




```
//Ο τύπος κόμβος
typedef struct NodeTag {
        InfoField Info;
        struct NodeTag *Link;
} NodeType;

// Ο τύπος δείκτη σε κόμβο
        typedef NodeType *NodePtr;
// Η δήλωση δείκτη αρχής της λίστας
        NodePtr L; // πάντα χρειάζεται μια αρχή
```

Εισαγωγή σε συνδεδεμένη λίστα (4 βήματα).

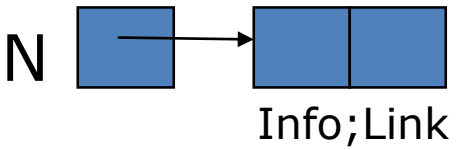
Περιπτώσεις:ΜΕΤΑ από πού; (αρχή, μέση, τέλος)



Τα αρχικά βήματα 1, 2 είναι κοινά σε κάθε περίπτωση

1. Δημιουργούμε νέο κόμβο N

```
NodePtr N; //δήλωση
N = malloc(sizeof(NodeType))
```



2. Βάζουμε τιμές στο Info του N

```
N->Info=…
```

Βήματα 3+4 για την σύνδεση του N (περιπτώσεις Α, Β)

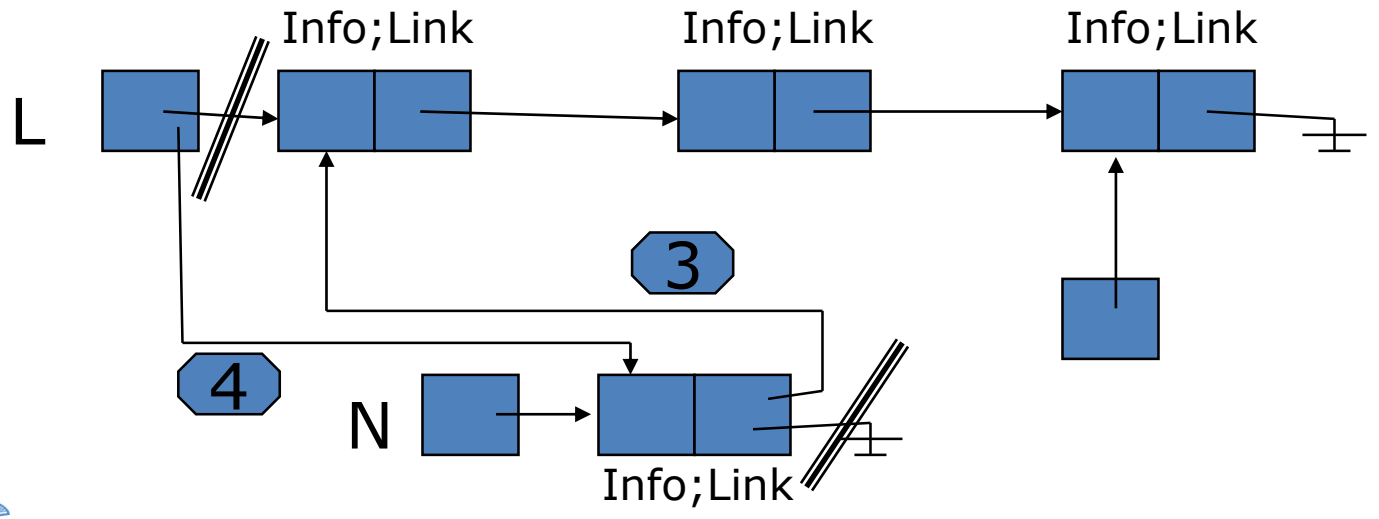
Α. Εισαγωγή στην αρχή (ισχύει και για κενή λίστα)

3. Τιμή στο Link του N

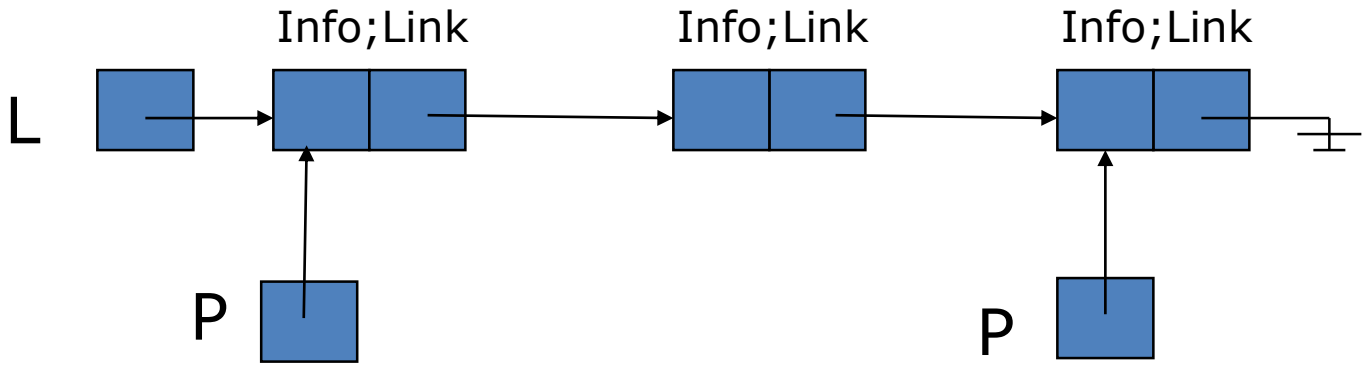
```
N->Link=L;//Ισχύει και αν η L==Κενή(L==NULL)
```

4. Τον κάνουμε τον πρώτο κόμβο της λίστας.

```
L=N;
```



Β. Εισαγωγή σε άλλη θέση (ΜΕΤΑ από προδείκτη P)

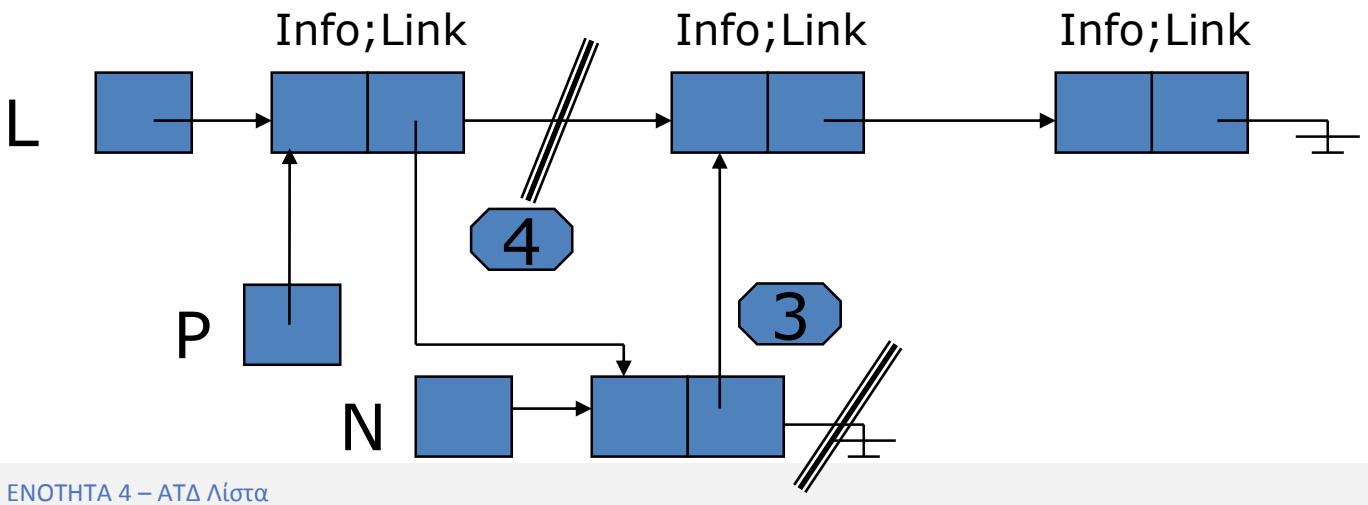


3. Τιμή στο Link του N

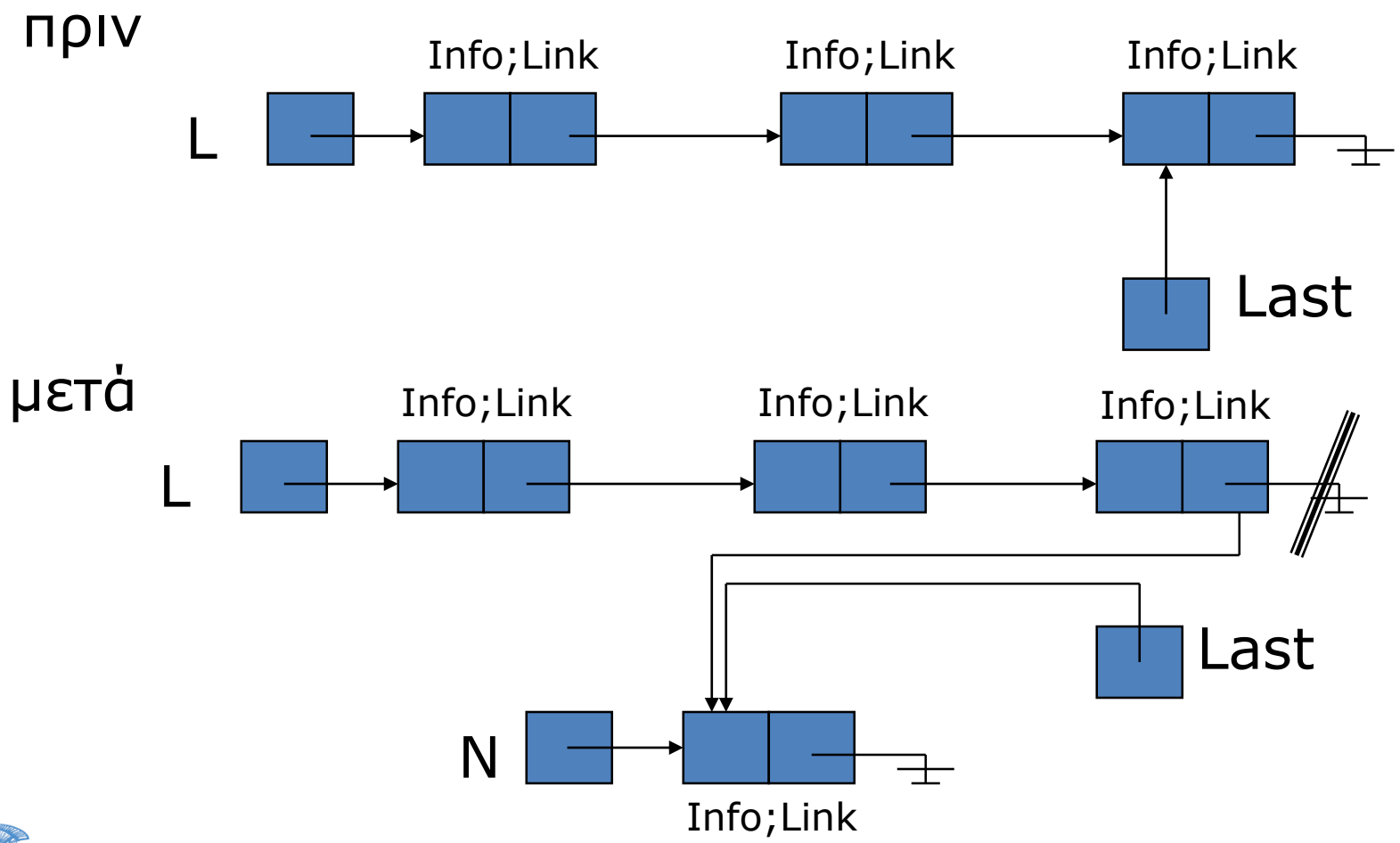
```
N->Link = P->Link;
```

4. Τον ενσωματώνουμε στη λίστα

```
P->Link = N;
```



Αν έχουμε δείκτη Last και κάνουμε εισαγωγή στο τέλος



… η Εισαγωγή στο τέλος (P==Last)
Βήματα 1, 2, 3, 4 όπως πριν + 5ο

1. Δημιουργούμε νέο κόμβο

```
NodePtr N;
N = malloc(sizeof(NodeType))
```

2. Βάζουμε τιμές στο Info του N

```
N->Info=….
```

3. Τιμή στο Link του N

```
N->Link = P->Link;   //ή N->Link=NULL;
```

4. Τον ενσωματώνουμε στη λίστα ως τελευταίο

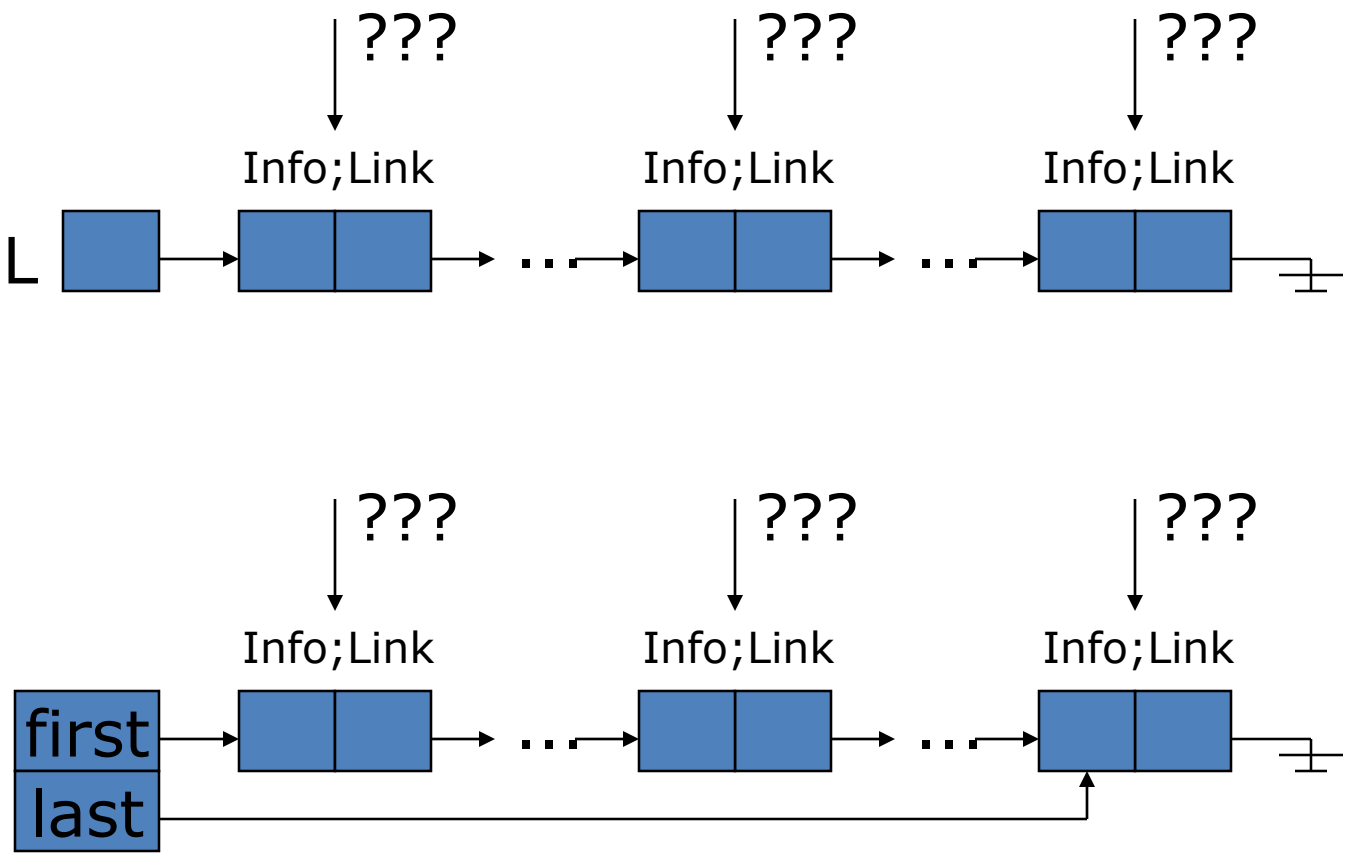
```
P->Link = N;          //ή Last->Link = N;
```

5. Έλεγχος και Ενημέρωση Δείκτη Last

```
if (P==Last)
    Last = N;
```

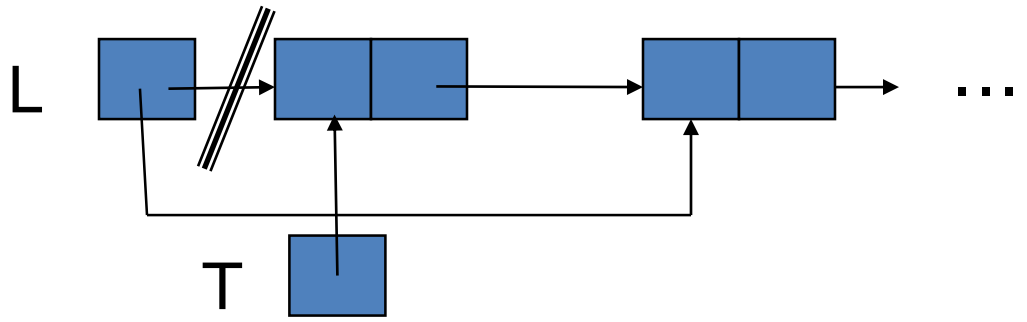
Είναι σωστό; Τι γίνεται αν ο τελευταίος είναι και ο πρώτος;

Διαγραφή Κόμβου (με/χωρίς Last)



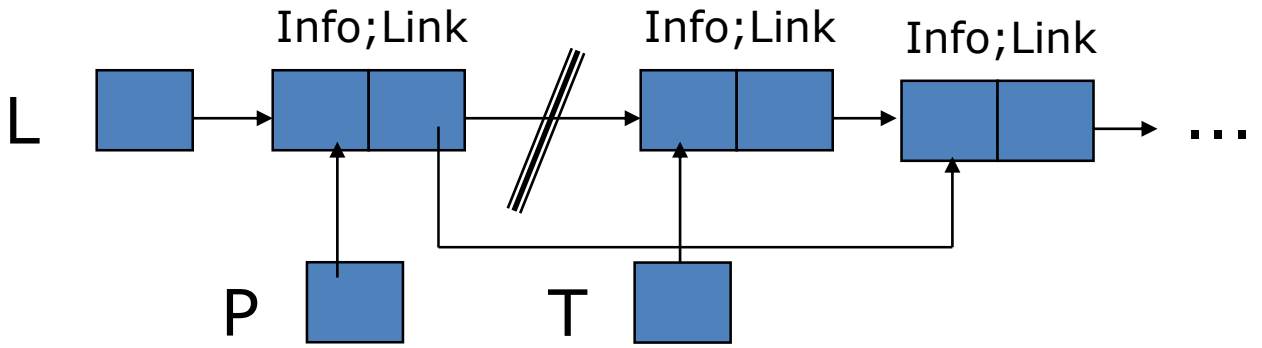
Διαγραφή του πρώτου κόμβου

```
typedef struct NodeTag {
        InfoField Info;
        struct NodeTag  *Link;
} NodeType;
```



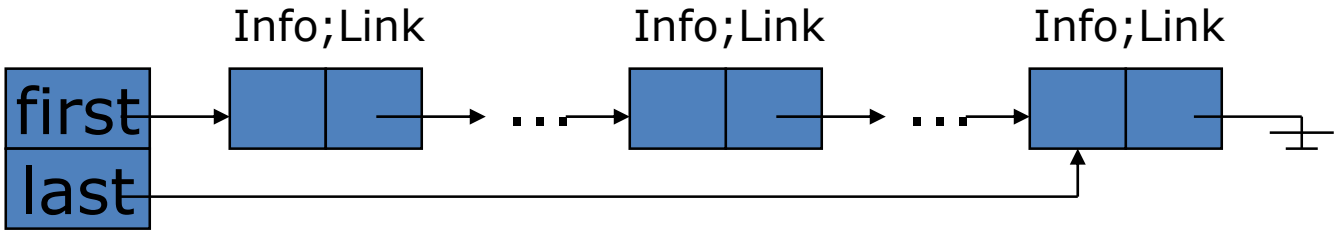

```
if (L!=NULL)          // υπάρχουν στοιχεία?
{     NodeType *T = L; // προσωρινός δείκτης για free
      L = L->Link;     // Νέα τιμή του L ο επόμενος
      if (L == NULL) // διαγραφή μοναδικού κόμβου
            Last = NULL; // τότε αλλάζει και ο Last
      free(T);         // ελευθέρωσε τον κόμβο
} else ;// ένδειξη εντοπισμού λάθους
```

Διαγραφή άλλου κόμβου (ΜΕΤΑ από τον προδείκτη P)



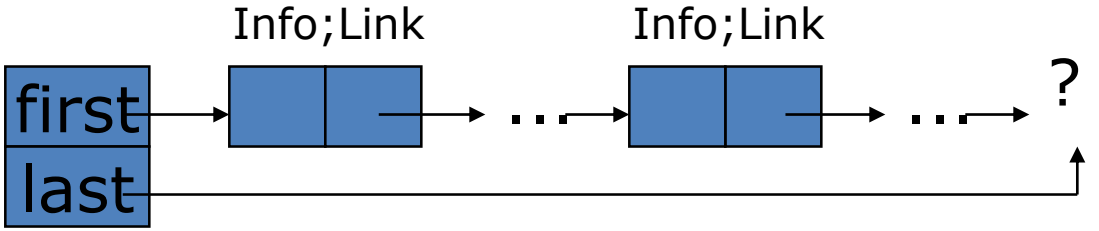

```
if ((P!=NULL)&&(P->Link!=NULL))// υπάρχει επόμενο;
{    NodeType *T=P->Link// προσωρινός δείκτης (free)
     P->Link = T->Link;// Νέα τιμή του L ο επόμενος
     if (T == Last)    // διαγραφή τελευταίου κόμβου;
          Last = P;
     free(T);          // ελευθέρωσε τον κόμβο
} else ;// ένδειξη εντοπισμού λάθους
```

Διαγραφή τελευταίου κόμβου ΠΑΝΤΑ με προδείκτη, όχι με χρήση του Last, παρόλο που δείχνει ήδη στον τελευταίο κόμβο.



Το free(Last) αφήνει έναν αιωρούμενο δείκτη και λάθος τιμή για το Last.

Μπορούμε να βρούμε τον προηγούμενο δείκτη P μόνο αν περάσουμε από P=first, μέχρι P->Link==Last.



Προτείνεται

1) Πριν τον προγραμματισμό συνδεδεμένων δομών, να σχεδιάζετε εικόνες για **κάθε** περίπτωση.
2) Απεικονίστε στους κόμβους τις μεταβλητές και τις τιμές τους. Σχεδιάστε σε διαδοχικά βήματα τις αλλαγές των δεικτών που θέλετε να κάνετε (εισαγωγή, διαγραφή, κλπ)
3) Διατηρείτε πάντα κάποια σύνδεση. Ένα συνηθισμένο λάθος είναι να ελευθερώνεται η μνήμη πριν φτιαχτούν οι συνδέσεις. Η σειρά ενημέρωσης των συνδέσεων είναι σημαντική.
4) Ελευθερώνετε την μνήμη που δεν χρειάζεστε.

Συνδεδεμένες Λίστες με stucts και pointers

Σχεδιασμός: Λίστα με Συνδεδεμένους Κόμβους

Κάθε στοιχείο του ΑΤΔ συνδεδεμένη λίστα (linked list) καλείται κόμβος (node) και περιέχει δύο πεδία.

- Στο ένα πεδίο αποθηκεύονται τα **δεδομένα** και
- Στο άλλο αποθηκεύεται η διεύθυνση του **επόμενου** κόμβου της λίστας.

Για καλύτερη απόκρυψη θα χρησιμοποιήσουμε ΟΑ και έναν αρχικό «πληροφοριακό» κόμβο της Λίστας (Δομής εν γένει).

Για κάθε λίστα, που το πρόγραμμά μας απαιτεί, δηλώνουμε ένα δείκτη ο οποίος περιέχει τη διεύθυνση του αρχικού της κόμβου.

1η Υλοποίηση του ΑΤΔ: μονά συνδεδεμένη λίστα με δείκτες

```
typedef ... TStoixeioyListas ;
typedef struct info_node * info_deikti;          //στο .h
-------------------------------------------------------
typedef struct info_node                        //στο .c για ΟΑ
{ int size;          /* το μέγεθος της λίστας */
  typos_deikti arxi;/*deikths ston proto komvo tis listas*/
} info_node;   /* o komvos plhroforias ths listas */


typedef struct typos_komvou *typos_deikti; //στο .h

typedef struct typos_komvou          //στο .c για ΟΑ
{ TStoixeioyListas dedomena;
  typos_deikti epomenos; // μονά συνδεδεμένη
};
-------------------------------------------------------
info_deikti lista1,lista2,…;//στο πρόγραμμα-πελάτης
```

Οι Πράξεις

```
info_deikti LIST_dimiourgia( );
void LIST_katastrofi(info_deikti * linfo);


int LIST_keni(const info_deikti linfo);


void LIST_epomenos(const info_deikti linfo,
                   typos_deikti * const p, int * const error);
void LIST_proigoymenos(const info_deikti linfo,
      typos_deikti * const p, int * const error);
void LIST_first(const info_deikti linfo,
      typos_deikti * const first, int * const error);
void LIST_last(const info_deikti linfo,
      typos_deikti * const last, int * const error);
```

```
/*prakseis prosbashs*/

void LIST_periexomeno(const info_deikti linfo,
   const typos_deikti p, TStoixeioyListas *val,
   int * const error);

void LIST_allagi(const info_deikti linfo,
   const typos_deikti p, TStoixeioyListas stoixeio,
   int * const error);

void LIST_diadromi(const info_deikti linfo,
   int * const error);


void LIST_eisagogi(const info_deikti linfo,
   TStoixeioyListas stoixeio,typos_deikti prodeiktis,
   int *error);

void LIST_diagrafi(const info_deikti linfo,
   typos_deikti *deiktis, int * const error);

void LIST_anazitisi(const info_deikti  linfo,
   TStoixeioyListas stoixeio, typos_deikti *prodeiktis,
   int *vrethike);
```

```
info_deikti LIST_dimiourgia( )

{/*   Pro:  kamia
*     Meta: Dhmioyrgia kenhs syndedemenhs listas meso
tis desmeusis kai arxikopoiisis tou  komvou linfo pou
leitourgei os komvos pliroforias kai tha sindethei me
ton proto komvo pou tha eisaxthei

*/

      info_deikti linfo;

      linfo = malloc(sizeof(info_node));

      linfo->size = 0;

      linfo->arxi = NULL;

      return linfo;

}
```

```
int LIST_keni(const info_deikti linfo)
{/*   Pro:  Dhmioyrgia Listas
  *   Meta: Epistrefei 1 an h lista einai kenh,
      diaforetika 0
*/

      return (linfo->arxi == NULL );
}
```

```
void LIST_epomenos(const info_deikti linfo,
      typos_deikti * const p, int * const error)
{/* Pro: Dhmiourgia listas
  * Meta:Epistrefei ton epomeno komvo tou p kai sto
error 0. An o p einai null tote epistrefei sto error
2 allios an den iparxei epomenos epistrefei sto
error 1 */

      *error = 0;
      if((*p)!=NULL)
       { if((*p)->epomenos!=NULL)
            *p=(*p)->epomenos;
        else
          *error=1;
      }
      else
        *error=2;
}
```

Η παράμετρος p είναι δείκτης σε μεταβλήτή τύπου δείκτη. Γιατί;

```
void LIST_periexomeno(const info_deikti linfo,
      const typos_deikti p, TStoixeioyListas *val,
      int * const error)
{/* Pro: O deikths p deixnei ena kombo sth lista
  * Meta:Epistrefei ta dedomena tou komvou tou p
*/

   *error=0;
    if(p!=NULL)
        TSlist_setValue(val, p->dedomena);
      else
         *error=1;
}
```

Χρησιμοποιούμε την TSlist_setValue(A,D) για να αποκρύψουμε την ανάθεση ("A = D ") για A, D οποιουδήποτε τυπου (π.χ. String). Η συνάρτηση ορίζεται στον Τύπο Στοιχείου (καλύτερος τρόπος μέσω παραμέτρου).

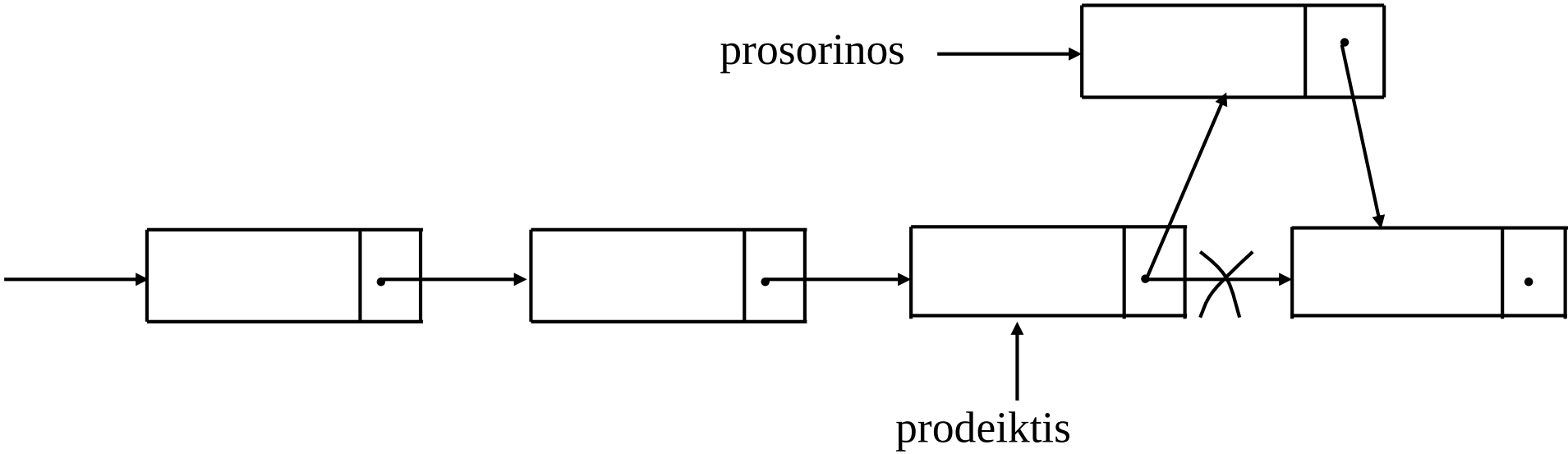
Η αλλαγή του περιεχομένου ενός κόμβου:

```
void LIST_allagi(const info_deikti linfo,
      const typos_deikti p,
      TStoixeioyListas stoixeio,
      int * const error)

{/*   Pro:  Dhmiourgia listas
  *   Meta: Allazei ta dedomena ston komvo toy p */
  *error=0;
  if(p!=NULL)
    TSlist_setValue(&(p->dedomena), stoixeio);
  else
    *error=1;
}
```

Παρατηρήστε την παράμετρο &(p->dedomena) στην κλήση της TSlist_setValue.

εισαγωγή και διαγραφή στοιχείου από μια συνδεδεμένη λίστα με δείκτες:

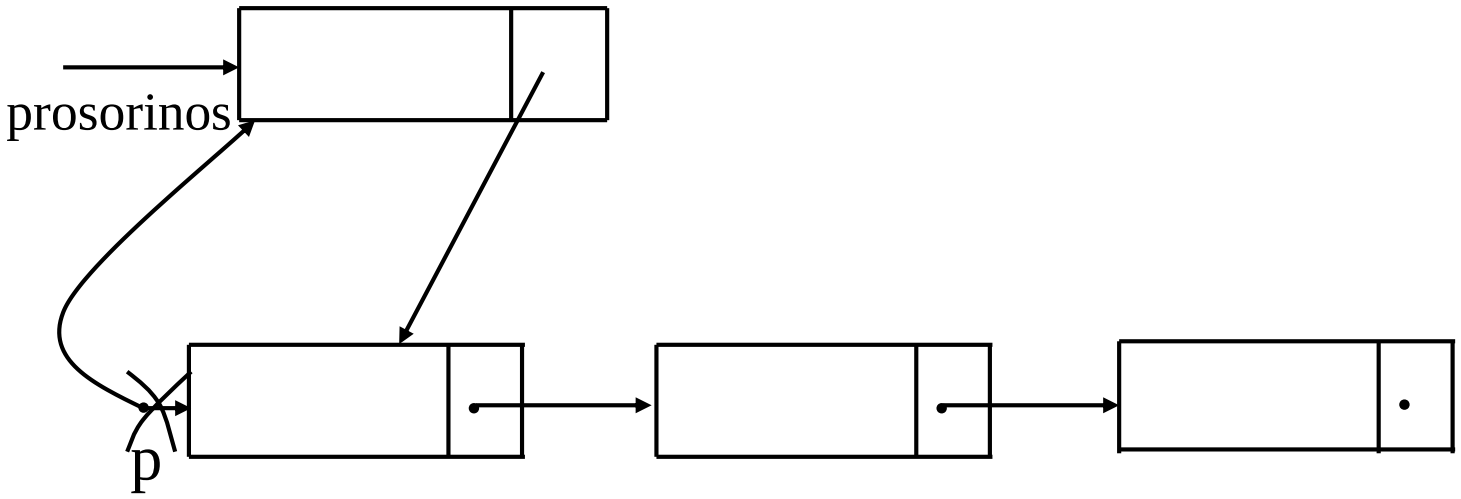
Εισαγωγή μετά από τον προδείκτη



```
void eisagogi_meta(const info_deikti linfo,
      TStoixeioyListas stoixeio,
      typos_deikti  prodeiktis, int *error)
{/* Pro: O prodeiktis deixnei ena kombo sth lista
  * Meta:O kombos me ta dedomena exei eisax8ei meta ton
           kombo pou deixnei o prodeiktis */

  typos_deikti prosorinos;                          //Βήμα 1
  prosorinos = malloc(sizeof(typos_komvou));
  if (prosorinos == NULL)
  {   *error=1;
      return;
  }
  TSlist_setValue(&prosorinos->dedomena, stoixeio);//2
  prosorinos->epomenos = prodeiktis->epomenos;     //3
  prodeiktis->epomenos = prosorinos;               //4

  linfo->size ++;              //ενημέρωση μεγέθους
}
```

Εισαγωγή Πριν

(χρειάζεται μόνο για εισαγωγή στην αρχή της λίστας)

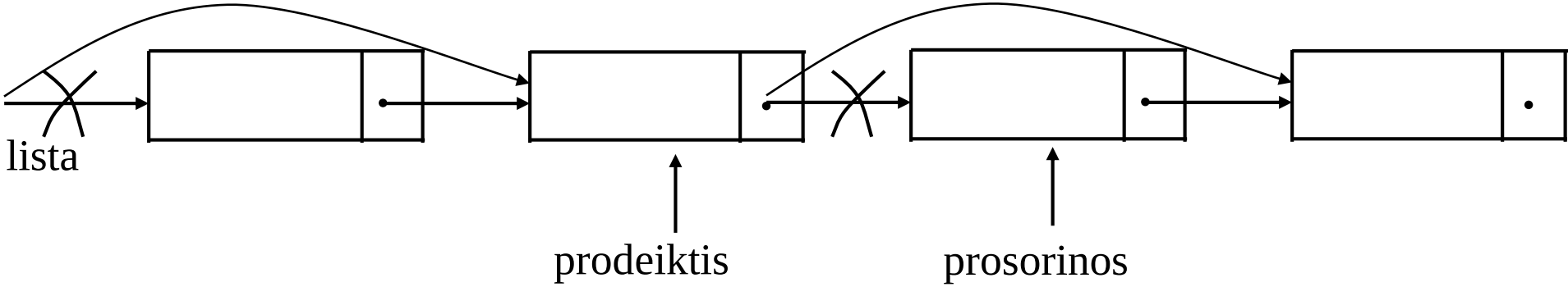


```
void eisagogi_arxi(const info_deikti linfo,
      TStoixeioyListas stoixeio, int *error)
{/* Pro: Dhmiourgia listas
  * Meta:O kombos me ta dedomena stoixeio exei eisax8ei
           sthn arxh ths listas */

  typos_deikti prosorinos;                          //Βήμα 1
  prosorinos = malloc(sizeof(typos_komvou));
  if ( prosorinos == NULL )
  {   *error=1;
      return;
  }

  TSlist_setValue(&prosorinos->dedomena,stoixeio); //2
  prosorinos->epomenos = linfo->arxi;               //3
  linfo->arxi = prosorinos;                         //4

  linfo->size ++;
}
```

Η εισαγωγή ενός κόμβου σε οποιοδήποτε σημείο της λίστας με σύμβαση η τιμή prodeiktis==NULL να σημαίνει «στην αρχή».

```
void LIST_eisagogi(const info_deikti linfo,
      TStoixeioyListas stoixeio,
      typos_deikti  prodeiktis, int *error)
{/* Pro: Dhmiourgia listas
  * Meta:Eisagetai to stoixeio meta ton prodeikti,an
      einai null mpainei stin arxi tis listas allios
      meta apo ton komvo pou deixnei*/

  *error=0;
  if(prodeiktis==NULL)
      eisagogi_arxi(linfo, stoixeio, error); //σύμβαση
  else
      eisagogi_meta(linfo,stoixeio, prodeiktis, error);

}
```

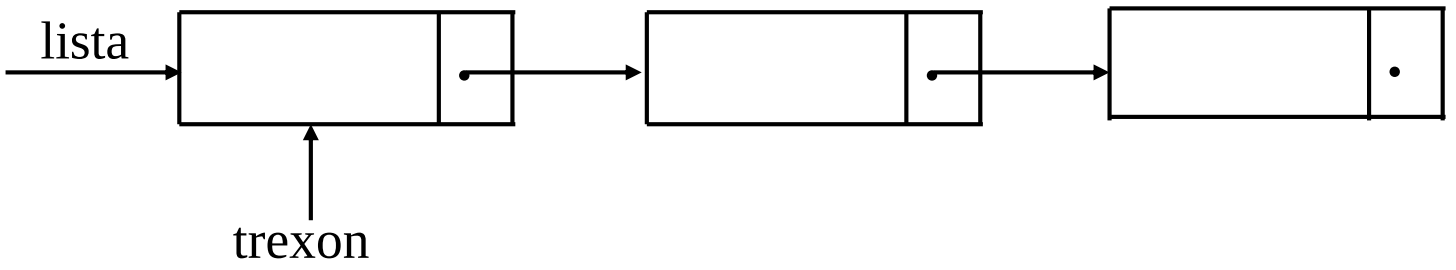
Διαγραφή



```
void LIST_diagrafi_prwtou(const info_deikti linfo,
                                  int * const error)
{/* Pro: Dhmiourgia Listas
  * Meta:Diagrafetai to prwto stoixeio ths listas  */

  typos_deikti prosorinos;

  *error=0;

  if LIST_keni(linfo)
  {   *error=1;
      return;
  }
  else
  {    prosorinos = linfo->arxi ;
       linfo->arxi  = prosorinos->epomenos;
       free(prosorinos);
      linfo-> size--;
  }
```

```
void LIST_diagrafi_meta(const info_deikti linfo,
      typos_deikti prodeiktis, int * const error)
{/* Pro: Dhmiourgia Listas
  * Meta:Diagrafetai to stoixeio ths listas meta ton
           prodeiktis */

{     if (prodeiktis->epomenos !=NULL){
           *error=0;
           typos_deikti prosorinos;

           prosorinos = prodeiktis->epomenos;
           prodeiktis->epomenos = prosorinos->epomenos;
           free(prosorinos);
           linfo->size--;
      }
      else *error=1; //ο προδείκτης είναι τελευταίος

}
```

```
void diagrafi_komvou(   const info_deikti linfo,
                        typos_deikti prodeiktis,
                        int *error){
/* Προ : Η *listaPtr δεν είναι κενή και ο prodeiktis
        είναι ένας κόμβος της λίστας ή είναι NULL.
   Μετά: Αν ο prodeiktis είναι NULL τότε διαγράφτηκε
        το πρώτο στοιχείο. Αν ο prodeiktis είναι κόμβος
        της λίστας τότε διαγράφτηκε ο επόμενος κόμβος */

        typos_deikti prosorinos;
        error = 0;

        if keni(linfo)
              *error=1;
        else if (prodeiktis == NULL)          // Σύμβαση
              diagrafi_prwtou(linfo, &error);
        else  diagrafi_meta(linfo, prodeiktis, &error);
}
```




```
void diadromi(typos_deikti lista){
/* Προ : Έχει δημιουργηθεί μια λίστα.
   Μετά: Έχει διατρέξει όλους τους κόμβους μιας λίστας
      και έχει επεξεργαστεί τα δεδομένα κάθε κόμβου. */

      typos_deikti trexon=lista;

      while (trexon!=NULL)
      {
            /* εντολές για επεξεργασία των δεδομένων */

            epomenos(lista, &trexon);
      }
}
```

```
void LIST_katastrofi(info_deikti * linfo)
{/* Pro: Dhmioyrgia listas
  * Meta:Katastrofi ths listas kai apodesmeusi   sto telos
kai tou komvou linfo */

    typos_deikti todel,todel2;
    todel=(*linfo)->arxi;

    while(todel!=NULL)
    {   todel2=todel;
        todel=todel->epomenos;
        free(todel2);
    }
    (*linfo)->arxi = NULL;
    free(*linfo);
    *linfo = NULL;
}
```

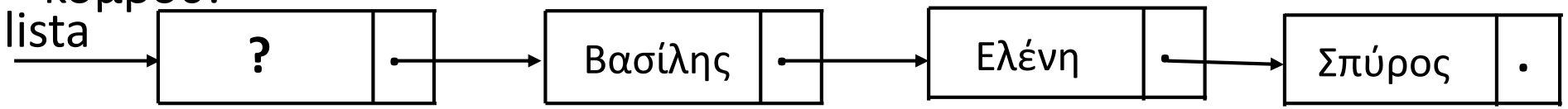
Άλλες μορφές Λιστών ίδια διεπαφή στο .h

- Με κεφαλή
- Κυκλικά Συνδεδεμένη
- Διπλά Συνδεδεμένη
- Με ένδειξη Τέλους (Last ή Tail)

- Πολλαπλά Συνδεδεμένες Λίστες

Λίστες με κεφαλή

Ο κόμβος κεφαλή συνήθως δεν περιέχει τίποτα στο τμήμα των δεδομένων του και ο ρόλος του είναι απλά για να υπάρχει ένας προηγούμενος κόμβος πριν τον "πραγματικό" πρώτο κόμβο της λίστας, ώστε να έχουμε 1 περίπτωση εισαγωγής/διαγραφής κόμβου.



Με αυτού του είδους την υλοποίηση, κάθε συνδεδεμένη λίστα έχει ένα κόμβο κεφαλή. Πιο συγκεκριμένα, μια κενή λίστα έχει ένα κόμβο κεφαλή:

Οι δηλώσεις δεν αλλάζουν, ΌΜΩΣ μερικές πράξεις ΝΑΙ

```
typedef struct info_node  *info_deikti;

typedef struct info_node
{ int size;
  typos_deikti arxi; /* ston proto komvo */
} info_node; /*o komvos plhroforias listas */

typedef struct typos_komvou *typos_deikti;

typedef struct typos_komvou{
     TStoixeioyListas dedomena;
     typos_deikti epomenos;
};
```

```
info_deikti LIST_dimiourgia()
{/* Pro: kamia
  * Meta:Dhmioyrgia kenhs syndedemenhs listas.
       desmeusis kai arxikopoiisis tou  komvou linfo
       (komvos pliroforias) kai tis desmeusis tou  protou
komvou tis listas pou poy leitourgei os     komvos kefali
tis listas */

  info_deikti linfo;
  linfo = malloc(sizeof(info_node));
  typos_deikti  head;
  head = malloc(sizeof(typos_komvou));
  head->epomenos = NULL;
  linfo->size = 0;
  linfo->arxi = head;
  return linfo;
}
```

Για τον έλεγχο αν μια λίστα με κεφαλή είναι κενή έχουμε:

```
int LIST_keni(const info_deikti linfo)
{/*  Pro: Dhmiourgia listas
  *  Meta:epistrefei 1 an h lista einai kenh,
          diaforetika 0 */


     return ( linfo->arxi->epomenos == NULL );
}
```

Οι πράξεις epomenos, periehomeno, allagh παραμένουν οι ίδιες.

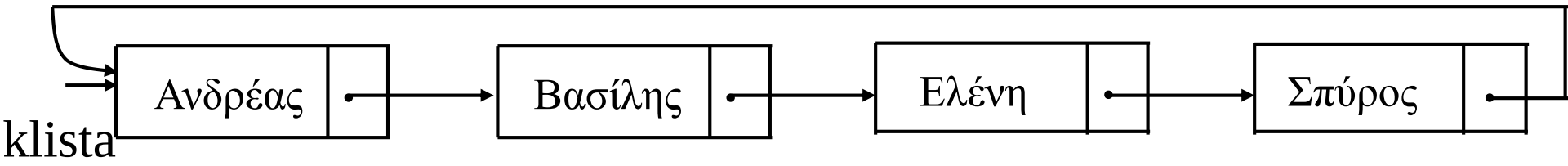
Μόνο μια πράξη για εισαγωγή και μια για διαγραφή

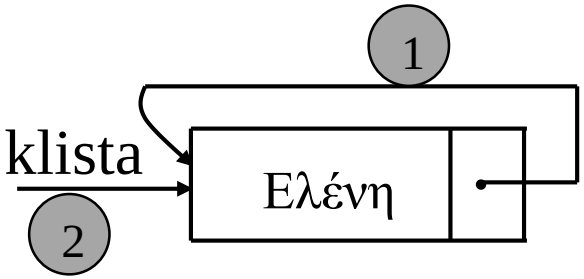
```
void LIST_eisagogi(const info_deikti linfo,
      TStoixeioyListas stoixeio,
            typos_deikti prodeiktis, int *error)
{/* Pro: Dhmiourgia listas
  * Meta:Eisagetai to "stoixeio" meta ton "prodeikti",an
      einai null autos to stoixeio mpainei stin arxi tis
      listas allios mpainei meta apo ton komvo pou
      deixnei autos */

  typos_deikti prosorinos;
  prosorinos = malloc(sizeof(typos_komvou));
  if ( prosorinos == NULL )
      {*error=1; return;}

  TSlist_setValue(&prosorinos->dedomena, stoixeio);
  prosorinos->epomenos = prodeiktis->epomenos;
  prodeiktis->epomenos = prosorinos;
  linfo->size ++;
}
```

Για το υποπρόγραμμα diagrafi έχουμε:

```
void diagrafi(const info_deikti linfo,
              typos_deikti prodeiktis, int *empty){
/* Προ : Η λίστα δεν είναι κενή.
   Μετά: διαγράφτηκε ο επόμενος κόμβος από αυτόν που
      δείχνει ο prodeiktis  */

      typos_deikti prosorinos;

      if (keni(lifo))
            *empty = 1;
      else {
            *empty=0;
            prosorinos = prodeiktis->epomenos;
            prodeiktis->epomenos = prosorinos->epomenos;
            free(prosorinos);
            linfo->size--;
      }
}
```

Κυκλικά Συνδεδεμένες Λίστες (π.χ. Επαφές στα κινητά)





Στους αλγορίθμους της διαγραφής και εισαγωγής κόμβου δεν χρειάζεται να ληφθεί υπόψη η περίπτωση κόμβων που δεν έχουν προηγούμενο.

Αλγόριθμος εισαγωγής σε κυκλικά συνδεδεμένη λίστα :

{Ο prodeiktis δείχνει τον προηγούμενο κόμβο από αυτόν που πρόκειται να εισαχθεί (αν υπάρχει)}

```
1. pare_neo_komvo(prosorinos)                          // Βήμα 1

2. dedomena(prosorinos)= stoixeio                          //2

3. a) Αν η κυκλική λίστα είναι κενή (νέα περίπτωση)
            epomenos(prosorinos) = prosorinos              //3
            klista = prosorinos                            //4
   b) διαφορετικά (όπως στην μη κυκλική)
            epomenos(prosorinos)=epomenos(prodeiktis)      //3
            epomenos(prodeiktis) = prosorinos              //4
```

```
void LIST_eisagogi(const info_deikti linfo,
                   TStoixeioyListas stoixeio,
                   typos_deikti prodeiktis, int *error)
{ typos_deikti prosorinos, temp;
  prosorinos = malloc(sizeof(typos_komvou));          //1
  TSlist_setValue(&(prosorinos->dedomena), stoixeio); //2
  if(prodeiktis == NULL) //gia eisagogi stin arxi
  {  if(LIST_keni(linfo))                          // NEO 3
           prosorinos->epomenos = prosorinos;
     else
     { LIST_last(linfo, &temp, error);    //prodeiktis
       prosorinos->epomenos = (*linfo)->arxi;
       temp->epomenos = prosorinos;
     }
     linfo->arxi= prosorinos;                      //4
  } else  //gia eisagogi meta ton prodeikti
  {     prosorinos->epomenos = prodeiktis->epomenos;
        prodeiktis->epomenos = prosorinos;
  }
  linfo->size ++;
}
```

Αλγόριθμος διαγραφής σε κυκλικά συνδεδεμένη λίστα

{Ο prodeiktis δείχνει τον προηγούμενο κόμβο από αυτόν που πρόκειται να διαγραφεί (αν υπάρχει)}

```
Αν η λίστα είναι κενή τότε λάθος
Διαφορετικά
```

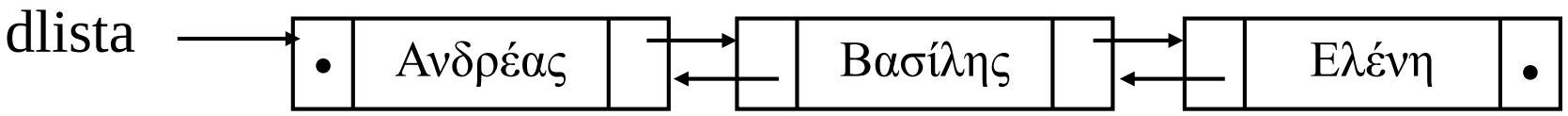
```
1. prosorinos = epomenos(prodeiktis)
2. Αν prosorinos == prodeiktis
   τότε {λίστα με ένα μόνο κόμβο}
        klista = NULL
   διαφορετικά
     epomenos(prodeiktis) = epomenos(prosorinos)
3. apodesmeysi(prosorinos)
```

```
void LIST_diagrafi(const info_deikti linfo,
                   typos_deikti *prodeiktis, int * const error)
{  typos_deikti prosorinos,previous;
   prosorinos = *prodeiktis;
   *error=0;
   if(LIST_keni(linfo)||(*deiktis == NULL))
   {   *error=1; return; }
   if(linfo->arxi->epomenos!=linfo->arxi)     // όχι MONO 1
   { LIST_proigoymenos(linfo, &previous, error);
     if(linfo->arxi==*prodeiktis) //an einai o protos kombos
        linfo->arxi=(*prodeiktis)->epomenos;
        *prodeiktis=prosorinos->epomenos;
        previous->epomenos = prosorinos->epomenos;
        free(prosorinos);
    }
    else //diagrafi tou monadikou kombou tis listas
    {  free(linfo->arxi);
       linfo->arxi = NULL;
       *prodeiktis = NULL;
    }
    linfo->size--;
}
```

Αλγόριθμος για την επίσκεψη των κόμβων μιας κυκλικά συνδεδεμένης λίστας

```
Αν η λίστα δεν είναι κενή τότε:

   1.trexon = klista

   2.Να επαναλαμβάνονται τα ακόλουθα βήματα:
      α. Επεξεργασία του dedomena(trexon)
      β. trexon = epomenos(trexon)
     μέχρις ότου trexon == klista
```

Διπλά Συνδεδεμένες Λίστες



Για την υλοποίηση των διπλά συνδεδεμένων λιστών χρειάζονται οι ακόλουθοι ορισμοί και δηλώσεις:

```
typedef struct typos_komvou
{   TStoixeioyListas dedomena;
     typos_deikti    epomenos;
     typos_deikti    proigoumenos;
} typos_komvou;
```

Ένας ακόμη δείκτης σε κάθε κόμβο.
Δεν αλλάζει το .h (εφόσον έχουμε τον αρχικό κόμβο πληροφορίας)

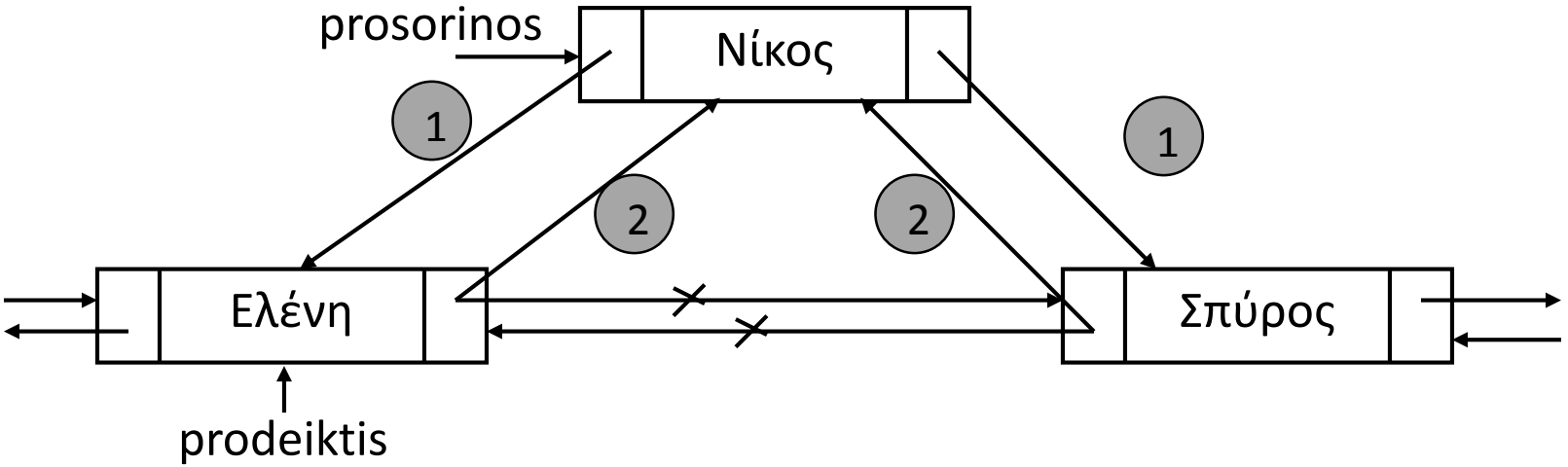
Οι αλγόριθμοι για τις βασικές πράξεις με διπλά συνδεδεμένες λίστες είναι όμοιοι με εκείνους των συνδεδεμένων λιστών μιας κατεύθυνσης. Διαχειριζόμαστε 2 δείκτες επιπλέον. Δεν έχουμε αλλαγές στη Δημιουργία.

```
info_deikti LIST_dimiourgia( )
{/* Pro:  kamia
  * Meta: Dhmioyrgia kenhs syndedemenhs listas meso tis
      desmeusis kai arxikopoiisis tou  komvou linfo pou
      leitourgei os komvos pliroforias kai tha sindethei
      me ton proto komvo pou tha eisaxthei */

      info_deikti linfo;
      linfo = malloc(sizeof(info_node));
      linfo->size = 0;
      linfo->arxi = NULL;
      return linfo;
}
```

Επομένως ούτε και στον έλεγχο κενής

```
int LIST_keni(const info_deikti  linfo)
{/* Pro: Dhmiourgia listas
  * Meta: epistrefei 1 an einai kenh, diaforetika 0 */

      return ( linfo->arxi == NULL );
}
```

Εισαγωγή κόμβου σε μια διπλά συνδεδεμένη λίστα με κεφαλή



Πιο συγκεκριμένα, πρέπει να ακολουθηθούν τα επόμενα βήματα:

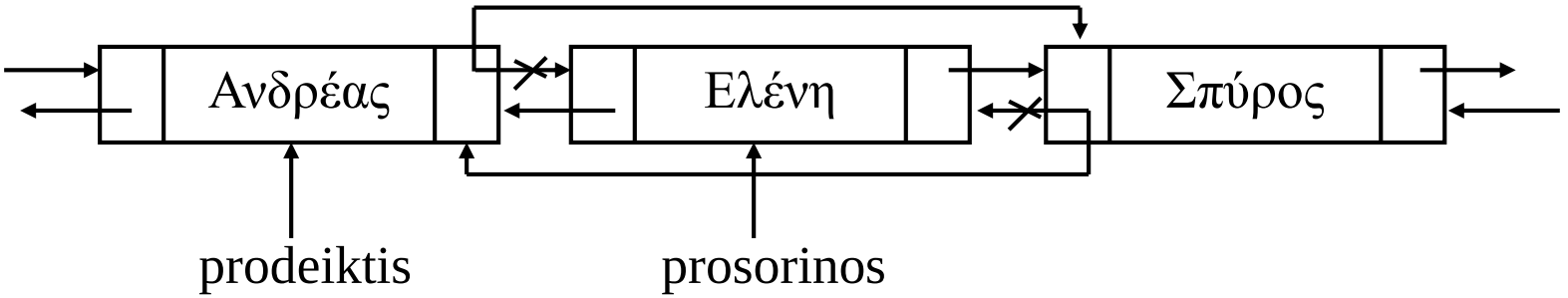
1. Δημιουργία κόμβου στον οποίο να δείχνει ο δείκτης prosorinos.//1
2. Καταχώρηση της τιμής στο τμήμα των δεδομένων του //2
3. Ενημέρωση δεικτών του κόμβου (2 δείκτες) //3+
4. Ενημέρωση δεικτών προηγούμενου και επόμενου (αν υπάρχει) //4+

```
void LIST_eisagogi(const info_deikti linfo,
                   TStoixeioyListas stoixeio,
                   typos_deikti prodeiktis, int *error)
{/* Pro: Dhmiourgia listas
  * Meta:Eisagetai to "stoixeio" meta ton "prodeikti",an
      einai null autos to stoixeio mpainei stin arxi tis
      listas allios mpainei meta apo ton komvo */

      *error=0;
      if(prodeiktis==NULL)
        eisagogi_arxi(linfo,stoixeio, error);
      else
        eisagogi_meta(linfo,stoixeio, prodeiktis, error);

}
```

```
void eisagogi_arxi(const info_deikti linfo,
            TStoixeioyListas stoixeio, int *error)
{/* Pro: Dhmiourgia listas
  * Meta:O kombos me ta dedomena stoixeio exei eisax8ei
       sthn arxh ths listas */

       typos_deikti prosorinos;
       prosorinos = malloc(sizeof(typos_komvou));
       if ( prosorinos == NULL )
       {   *error=1;
             return;
       }
       TSlist_setValue(&prosorinos->dedomena, stoixeio);
       prosorinos->epomenos = linfo->arxi;
       prosorinos->proigoumenos = NULL;
       if (linfo->arxi != NULL)
             linfo->arxi->proigoumenos = prosorinos;
       linfo->arxi = prosorinos;
       linfo->size ++;
}
```

```
void eisagogi_meta(const info_deikti linfo,
                   TStoixeioyListas stoixeio,
                   typos_deikti prodeiktis, int *error)
{ /* Pro: Dhmioyrgia listas
   * Meta:O kombos me ta dedomena stoixeio eisagetai
   *        meta ton kombo pou deixnei o prodeikths*/

      typos_deikti prosorinos;
      prosorinos = malloc(sizeof(typos_komvou));

      TSlist_setValue(&(prosorinos->dedomena), stoixeio);

      prosorinos->epomenos = prodeiktis->epomenos;
      prosorinos->proigoumenos = prodeiktis;
      if(prosorinos->epomenos!=NULL)
          prosorinos->epomenos->proigoumenos=prosorinos;
      prodeiktis->epomenos = prosorinos;
      linfo->size ++;
}
```

Διαγραφή κόμβου από μια διπλά συνδεδεμένη λίστα με κεφαλή



```
LIST_diagrafi(const info_deikti linfo,
      typos_deikti *deiktis, int * const error)
{     typos_deikti prosorinos, previous;
      *error=0;
      if (LIST_keni(*linfo)||(*deiktis==NULL))
      { *error=1; return; }
      if(linfo->arxi==*deiktis)  //an einai o protos
      {   prosorinos = *deiktis;
          *deiktis=prosorinos->epomenos;
          if((*deiktis)!=NULL)
                  (*deiktis)->proigoumenos = NULL;
          linfo->arxi=prosorinos->epomenos;
      } else                      //se kathe alli periptosi
      {  prosorinos = *deiktis;
         previous=prosorinos->proigoumenos;
         *deiktis=prosorinos->epomenos;
         previous->epomenos=prosorinos->epomenos;
         if((*deiktis)!=NULL)
                  (*deiktis)->proigoumenos = previous;
      } free(prosorinos);
        linfo->size--;
```

Λίστα με δείκτη στο τέλος

Αλλάζουμε τον κόμβο πληροφορίας, όχι το .h

```
typedef struct info_node
{     int size;
      typos_deikti arxi;
            /* deikths sto proto komvo tis listas */
      typos_deikti telos;
            /*deikths sto teleutaio komvo tis listas*/
} info_node;    /* o komvos plhroforias ths listas */
```

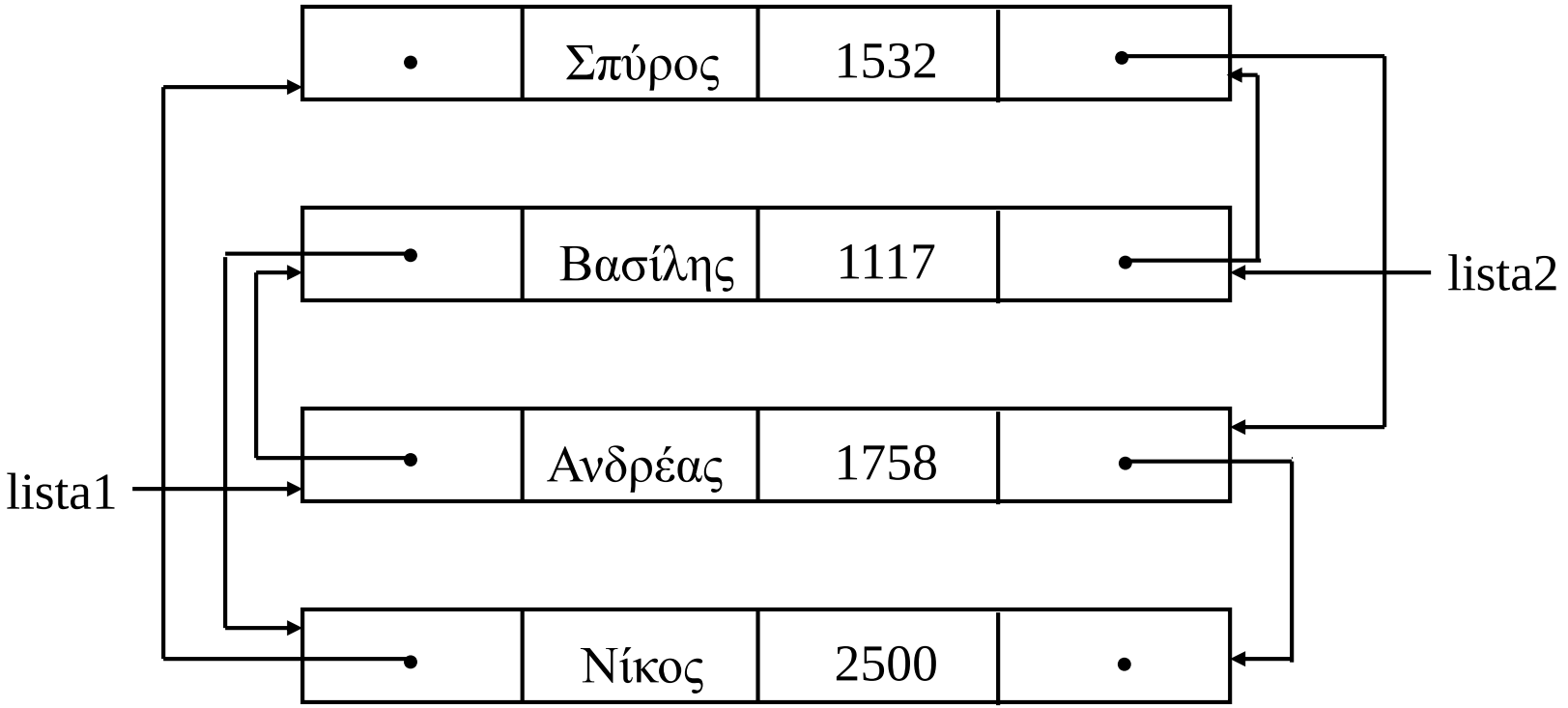
```
info_deikti LIST_dimiourgia( )
{/*   Pro:  kamia
  *   Meta: Dhmioyrgia kenhs syndedemenhs listas meso
            tis desmeusis kai arxikopoiisis tou  komvou
            linfo pou leitourgei os komvos pliroforias
            kai tha sindethei me ton proto komvo pou tha
            eisaxthei */

    info_deikti linfo;
    linfo = malloc(sizeof(info_node));
    linfo->size = 0;
    linfo->arxi = NULL;
    linfo->telos = NULL;
    return linfo;
}
```

```
void eisagogi_arxi(const info_deikti linfo,
            TStoixeioyListas stoixeio, int *error)
{/*   Pro: Dhmiourgia listas
  *   Meta:O kombos me ta dedomena stoixeio exei
            eisax8ei sthn arxh ths listas */
   int error=0;
   typos_deikti prosorinos;
   typos_deikti last=NULL;
   prosorinos = malloc(sizeof(typos_komvou));
   if ( prosorinos == NULL )
      {   *error=1; return;}

   TSlist_setValue(&prosorinos->dedomena, stoixeio);
   prosorinos->epomenos = linfo->arxi;
   linfo->arxi = prosorinos;
   linfo->size ++;

   LIST_last(linfo,&(linfo->telos), &error);
}
```

```
void eisagogi_meta(const info_deikti linfo,
                   TStoixeioyListas stoixeio,
                   typos_deikti prodeiktis, int *error)
{/* Pro: O prodeikths deixnei ena kombo sth lista
  * Meta:O kombos exei eisax8ei meta ton prodeikths */

      int error=0;
      typos_deikti prosorinos;

      prosorinos = malloc(sizeof(typos_komvou));
      if (prosorinos == NULL)
      {   *error=1;
            return;
      }
      TSlist_setValue(&prosorinos->dedomena, stoixeio);
      prosorinos->epomenos = prodeiktis->epomenos;
      prodeiktis->epomenos = prosorinos;
      linfo->size ++;
      LIST_last(*linfo,&(linfo->telos), &error)
}
```

Πολλαπλά συνδεδεμένες λίστες (multiply linked lists)

Για το προηγούμενο παράδειγμα των εγγραφών, που περιέχουν το όνομα και τον αριθμό μητρώου φοιτητών, θα μπορούσαμε να έχουμε την παρακάτω συνδεδεμένη λίστα (με δύο διασυνδέσεις):



Ειδική Περίπτωση

Ταξινομημένη Λίστα

Μια λίστα λέγεται **ταξινομημένη (sorted list)** αν οι κόμβοι της είναι συνδεδεμένοι κατά τέτοιο τρόπο ώστε ένα από τα πεδία, το **πεδίο κλειδί (key field)** του τμήματος δεδομένων, είναι ταξινομημένο σε αύξουσα ή φθίνουσα σειρά.

Σε μια τέτοια λίστα η εισαγωγή ενός νέου κόμβου θα πρέπει να έχει σαν αποτέλεσμα την παραγωγή μιας νέας ταξινομημένης λίστας. Η εισαγωγή του δηλαδή θα πρέπει να γίνει σε κάποιο συγκεκριμένο σημείο της λίστας.

Στην περίπτωση της αναζήτησης κάποιου κόμβου σε μια ταξινομημένη λίστα εξετάζεται επιπλέον αν προσπεράσαμε την τιμή που αναζητούμε.

Αλγόριθμος αναζήτησης ταξινομημένης (σε αύξουσα σειρά) λίστας (ο prodeiktis για τυχόν εισαγωγή ή διαγραφή)

1. prodeiktis = null, trexon = list

2. Όσο δεν ολοκληρώθηκε η αναζήτηση και trexon != null να εκτελούνται οι παρακάτω εργασίες:

    Αν dedomena(trexon) >= stoixeio, τότε

        Η αναζήτηση ολοκληρώθηκε

        αν dedomena(trexon) = = stoixeio ο κόμβος βρέθηκε διαφορετικά

        προχώρησε τους δείκτες prodeiktis και trexon.

Η υλοποίηση του παραπάνω αλγορίθμου γίνεται με την ακόλουθη διαδικασία :

```
void anazitisi(typos_deikti lista,
               TStoixeioyListas stoixeio,
               typos_deikti *prodeiktis, int *vrethike){

/* Προ : Τα στοιχεία της λίστας lista είναι
        ταξινομημένα σε αύξουσα σειρά.
   Μετά: Έγινε αναζήτηση στη lista για τον εντοπισμό
        κόμβου που έχει περιεχόμενο stoixeio
        (*vrethike==1) ή για μια θέση για την εισαγωγή
        νέου κόμβου (*vrethike==0).
        Ο *prodeiktis δείχνει τον προηγούμενο κόμβο από
        αυτόν που περιέχει το stoixeio(εφόσον βρέθηκε) ή
        τον κόμβο μετά τον οποίο μπορεί να εισαχθεί το
        stoixeio (εφόσον δεν βρέθηκε).
*/
```

συνέχεια

```
    typos_deikti trexon;
    int anazitisi=0;
    trexon = lista;
    *prodeiktis = NULL;
    *vrethike = 0;
    while ((!anazitisi) && (!keni(trexon)))
        if (periexomeno(trexon) >= stoixeio){
            anazitisi = 1; // τέλος αναζήτησης
            *vrethike =
                (periexomeno(trexon)==stoixeio);
        }
        else{
            *prodeiktis = trexon;
            proxorise(lista, &trexon);
        }
}
```

Εφαρμογές συνδεδεμένων λιστών

Παράσταση πολυωνύμου με συνδεδεμένη λίστα

Ενα πολυώνυμο n-ιοστού βαθμού έχει τη μορφή:

$$p_n(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

και μπορεί να μελετηθεί με τη χρήση του ΑΤΔ λίστα. Πράγματι, οι συντελεστές του μπορεί να θεωρηθούν ότι σχηματίζουν τη λίστα

$$(a_0, a_1, a_2, \ldots , a_n)$$

και κατά συνέπεια να παρασταθεί με οποιαδήποτε από τις υλοποιήσεις για λίστα. Σε περίπτωση που το πολυώνυμο είναι αραιό π.χ το :

$$p(x) = 6 + x^{50} + x^{99}$$

το οποίο έχει τη πλήρη μορφή

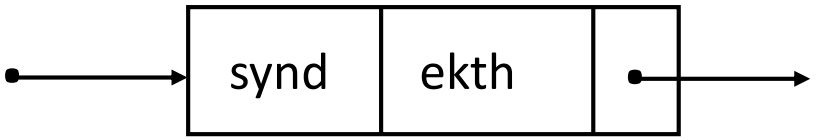
$$p(x) = 6 + 0x + 0x^2 + \ldots + 1x^{50} + \ldots + 1x^{99}$$

θα χρειάζεται ένα πίνακα με 100 στοιχεία, από τα οποία τα τρία είναι μη μηδενικά. Για την αποφυγή του προβλήματος αυτού θα πρέπει να διατηρούνται μόνο οι μη μηδενικοί συντελεστές του πολυωνύμου.

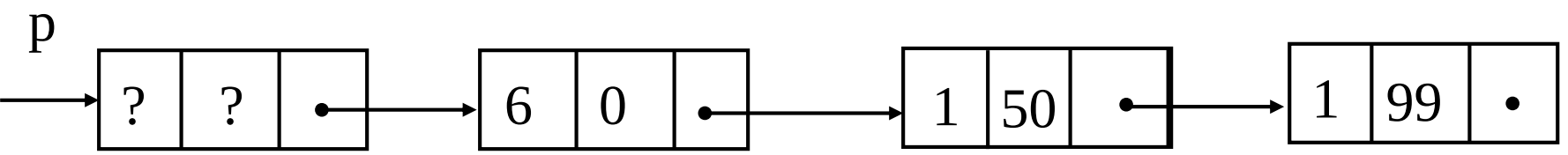
Για παράδειγμα το p(x) παριστάνεται σαν

((6, 0), (1, 50), (1, 99))

όπου τα ζευγάρια είναι διατεταγμένα έτσι ώστε οι εκθέτες να είναι σε αύξουσα σειρά. Ο κόμβος μιας τέτοιας λίστας θα έχει τη μορφή



Για παράδειγμα, το p(x) θα παριστάνεται με την ακόλουθη συνδεδεμένη λίστα με κεφαλή:



$$p(x) = 6 + 0x + 0x^2 + \ldots + 1x^{50} + \ldots + 1x^{99}$$

((6, 0), (1, 50), (1, 99))

Χρησιμοποιώντας την υλοποίηση με δείκτες, απαιτούνται οι ακόλουθοι ορισμοί και δηλώσεις:

```
typedef struct PolyElement {
      float synd;
      int ekth;
} TStoixeioyListas ;

typedef  struct typos_komvou *typos_deikti;
typedef struct k {
      TStoixeioyListas    dedomena;
      typos_deikti        epomenos;
} typos_komvou;
```

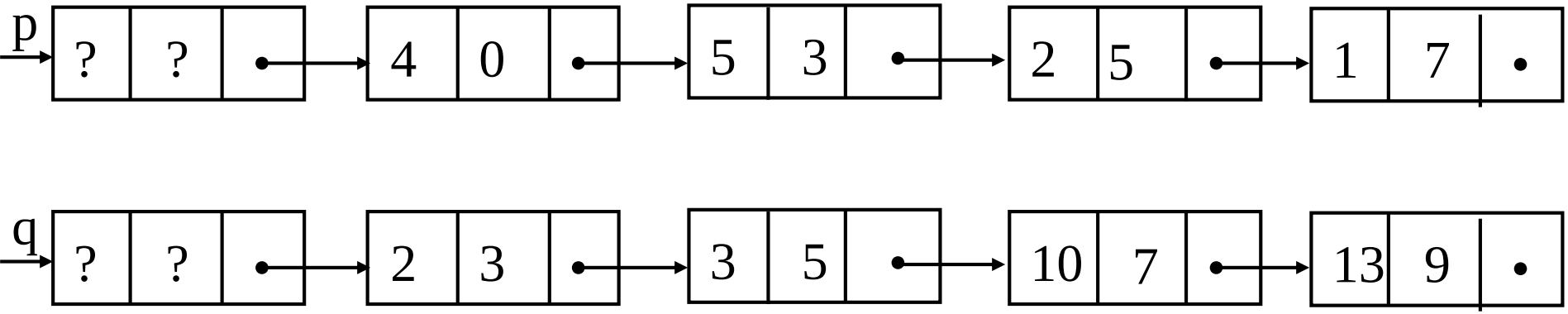
Η επεξεργασία πολυωνύμων που παριστάνονται με τον παραπάνω τρόπο δεν παρουσιάζει δυσκολίες. Για παράδειγμα, ας υποτεθεί ότι επιθυμούμε να προσθέσουμε τα πολυώνυμα

$$p(x) = 4 + 5x^3 + 2x^5 + x^7$$

και

$$q(x) = 2x^3 + 3x^5 + 10x^7 + 13x^9$$

που παριστάνονται ως (λίστα με κεφαλή)



Δύο βοηθητικοί δείκτες ptrexon και qtrexon διατρέχουν τα στοιχεία των λιστών p και q, αντίστοιχα. Νέα λίστα r (p+q).

Σύγκριση των εκθετών των δύο κόμβων των ptrexon και qtrexon.
Εκθέτες είναι διαφορετικοί

Ο κόμβος που περιέχει τον μικρότερο τοποθετείται στη r
προχωράμε αντίστοιχο δείκτη

Εκθέτες είναι ίσοι

οι συντελεστές προστίθενται και αν το άθροισμα δεν είναι μηδέν δημιουργείται νέος κόμβος. Το αποτέλεσμα τοποθετείται στο πεδίο synd. Ο κοινός εκθέτης στο πεδίο ekth. Ο κόμβος αυτός τοποθετείται στη λίστα r.

Προχωράμε και τους δύο δείκτες

Αν συναντήσουμε το τέλος της μιας λίστας, τότε αντιγράφονται οι υπόλοιποι κόμβοι της άλλης στη λίστα r.

Η πρόσθεση δύο πολυωνύμων που παριστάνονται με συνδεδεμένες λίστες υλοποιείται στο ακόλουθο υποπρόγραμμα:

```
void prosthesi_poly(info_deikti p,
                    info_deikti q,
                    info_deikti *rPtr)
{

/* Προ : Έχουν δημιουργηθεί δύο λίστες με
       κεφαλές που παριστάνουν δύο πολυώνυμα p
       και q.
   Μετά: Το *rPtr είναι δείκτης σε μια
       συνδεδεμένη λίστα (με κεφαλή) που
       παριστάνει το άθροισμα των πολυωνύμων p
       και q.
*/
```

συνέχεια

```
typos_deikti ptrexon, qtrexon, rtrexon, prosorinos;
TStoixeioyListas neos;
int error = 0;

LIST_first(p,&ptrexon, &error );// στον 1ο κόμβο
LIST_first(q,&qtrexon, &error); // στον 1οκόμβο
(*r)=LIST_dimiourgia();          // λίστα κενή

while ((ptrexon!=NULL) && (qtrexon!=NULL)){
 if (ptrexon->dedomena.ekth < qtrexon->dedomena.ekth){
     /* εκθέτης του p μικρότερος του εκθέτη του q */
     LIST_eisagogi(rPtr, ptrexon->dedomena, &rtrexon);
     LIST_epomenos(&ptrexon);
 }else
 if (qtrexon->dedomena.ekth < ptrexon->dedomena.ekth){
     /* εκθέτης του q μικρότερος του εκθέτη του p */
     LIST_eisagogi(rPtr, qtrexon->dedomena, &rtrexon);
     LIST_epomenos(&qtrexon);
 } else /* εκθέτες ίσοι */
```

συνέχεια

```
        /* εκθέτες ίσοι */
           neos.ekth = ptrexon->dedomena.ekth;

           neos.synd =  ptrexon->dedomena.synd
                      + qtrexon->dedomena.synd;

           if (neos.synd != 0)
                LIST_eisagogi(rPtr, neos, &rtrexon);


           LIST_epomenos(&ptrexon);

           LIST_epomenos(&qtrexon);

        }

         LIST_last(&rtrexon);

} /* end of while pointer ptrexon or qtrexon not null */
```

συνέχεια

```
/* Αντιγραφή υπόλοιπων κόμβων της p ή της q στην r */
    if (ptrexon!=NULL)
              prosorinos = ptrexon;
   else       prosorinos = qtrexon;

   while (prosorinos!=NULL){
      LIST_eisagogi(rPtr,prosorinos->dedomena,&rtrexon);
      LIST_epomenos(&prosorinos);
      LIST_last(&rtrexon);
   }
}
```

Υλοποίηση του ΑΤΔ στοίβα με συνδεδεμένη λίστα

Η στοίβα υλοποιείται ως μια συνδεδεμένη λίστα, όπου ο δείκτης που δείχνει τον πρώτο κόμβο της λίστας θα παίζει τον ρόλο της korifi.

Απαιτείται η εισαγωγή και διαγραφή κόμβων μόνο στην αρχή. Επομένως μονά συνδεδεμένη χωρίς κεφαλή είναι ικανοποιητική.

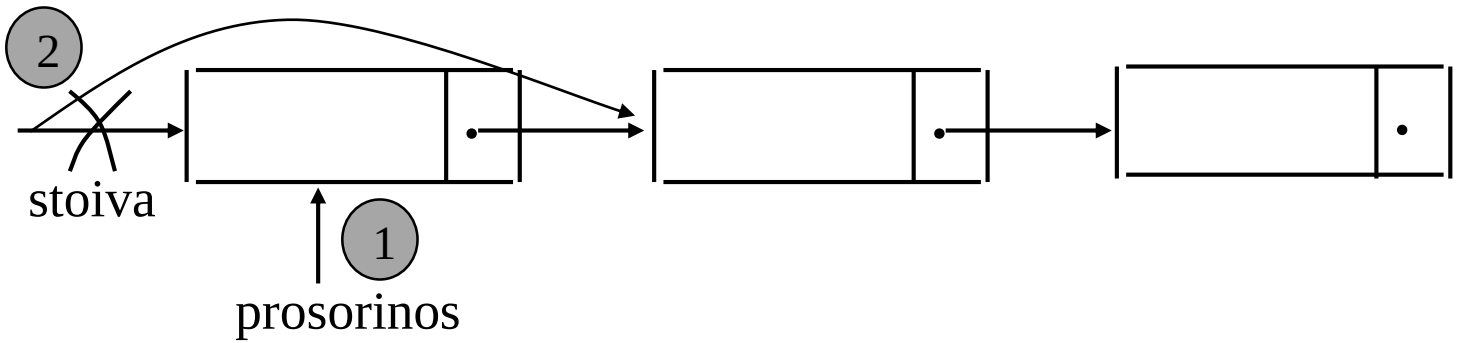
Για την υλοποίηση του ΑΤΔ στοίβα με συνδεδεμένες λίστες χρειάζονται οι δηλώσεις:

```
typedef ... typos_stoixeiou;
typedef struct typos_komvou *typos_stoivas;

typedef struct typos_komvou {
      typos_stoixeiou dedomena;
      typos_komvou * epomenos;
};
```

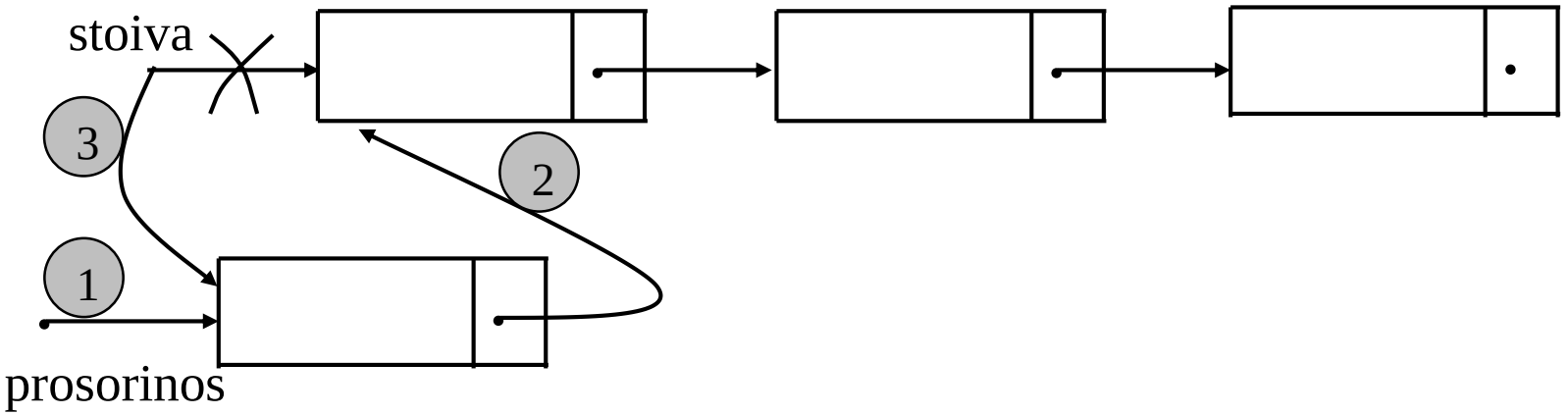
Η εξαγωγή και η ώθηση στοιχείου από τη στοίβα είναι ειδικές περιπτώσεις εισαγωγής και εξαγωγής στοιχείου από μια συνδεδεμένη λίστα (στην αρχή της λίστας).

Στη συνέχεια παρουσιάζονται τα υποπρογράμματα για τις βασικές λειτουργίες ώθησης και εξαγωγής στοιχείου σε μια στοίβα.

Η εξαγωγή (διαγραφή πρώτου κόμβου)



Η ώθηση (εισαγωγή πρώτου κόμβου)



Υλοποίηση ΑΤΔ Στοίβα ως Συνδεδεμένη Λίστα Με Μερική Απόκρυψη (δήλωση τύπου στο .h)

```
typedef struct ListStruct * TStoivas;
struct ListStruct
{ TStoixeioyStoivas dedomena;   /*to dedomeno*/
  ListStruct * epomenos; /*deikths ston epomeno komvo */
};


TStoivas Stoiva_dimiourgia( );
int Stoiva_keni(TStoivas stoiva);
void Stoiva_exagogi(TStoivas *stoivaPtr,
      TStoixeioyStoivas *stoixeio, int *ypoxeilisi);
void Stoiva_othisi(TStoivas *stoivaPtr,
      TStoixeioyStoivas stoixeio, int *yperxeilisi);
```

Υλοποίηση ΑΤΔ Στοίβα με Συνδεδεμένη Λίστα Με Ολική Απόκρυψη (δήλωση κόμβου στο .c)

```
typedef struct ListStruct *TStoivas;       /* δηλώσεις στο .h */


TStoivas Stoiva_dimiourgia();

void Stoiva_Destructor(TStoivas *StoivaPtr);

int  Stoiva_keni(TStoivas stoiva);

void Stoiva_exagogi(TStoivas *stoivaPtr,
              TStoixeioyStoivas *stoixeio, int *ypoxeilisi);

void Stoiva_othisi(TStoivas stoivaPtr,
       TStoixeioyStoivas stoixeio, int *yperxeilisi);
```

```
struct ListStruct{          /* struct στο .c */
       TStoixeioyStoivas dedomena;       /*to dedomeno*/
       ListStruct * epomenos; /*deikths ston epomeno komvo */
};
```

Υλοποιήσεις Πράξεων

```
TStoivas Stoiva_dimiourgia( ){
  return NULL;
}

int Stoiva_keni(TStoivas s){
  return (s == NULL);
}
```

Ώθηση (εισαγωγή στην αρχή)

```
void Stoiva_othisi(TStoivas *stoivaPtr,
      typos_stoixeiou stoixeio, int * error){

     struct ListStruct * prosorinos;
    *error = 0;
     prosorinos = malloc(sizeof(typos_komvou));// Βήμα 1
     if (prosorinos != NULL) {// συνθήκη «γεμάτη»
            prosorinos->dedomena = stoixeio;  // Βήμα 2
            prosorinos->epomenos = *stoivaPtr;// Βήμα 3
            *stoivaPtr = prosorinos;      // Βήμα 4
   } else
      *error = 1; /* η ώθηση δεν έγινε */
}
```

Στη πράξη othisi δεν χρησιμοποιείται η πράξη «γεμάτη» (όπως στην υλοποίηση με πίνακα) αλλά μόνο ο έλεγχος για το αν υπάρχει χώρος. Η πράξη «γεμάτη» δεν έχει νόημα για την υλοποίηση της στοίβας με συνδεδεμένη λίστα, ενώ μας φαινόταν αποδεκτή για την υλοποίηση με πίνακα.

Ο μόνος περιορισμός στην υλοποίηση της στοίβας με δείκτες είναι η συνολική διαθέσιμη μνήμη.

Ένα ερώτημα που (πρέπει να) θέτουμε στον σχεδιασμό μιας διεπαφής είναι: «Είναι γενικός σχεδιασμός ή μήπως (άμεσα – έμμεσα) εξαρτάται από την υλοποίηση;»

Εξαγωγή (διαγραφή πρώτου)

```
void exagogi(Stoiva *stoivaPtr,
        TStoixeioyStoivas *stoixeio, int *ypoxeilisi)

{ struct ListStruct * temp;
   if (!keni(*s) {
        *ypoxeilisi = 0;
        *stoixeio = (*stoivaPtr)->dedomena; //setValue
        temp=*stoivaPtr;
        *stoivaPtr = temp->epomenos;
        free(temp);
   }
   else *ypoxeilisi = 1;
}
```

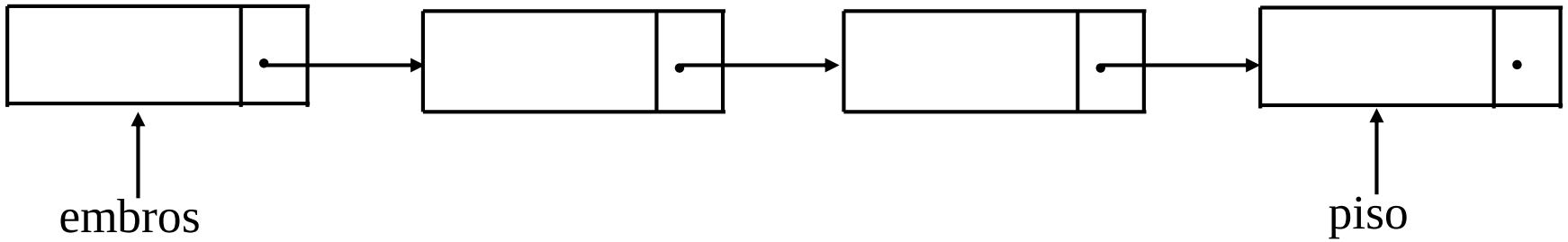
Ας σημειωθεί ότι στην Ολική Απόκρυψη οι επικεφαλίδες των υποπρογραμμάτων που υλοποιούν τις βασικές πράξεις του ΑΤΔ στοίβα με συνδεδεμένη λίστα είναι οι ίδιες με εκείνες που υλοποιούν τον ίδιο ΑΤΔ με πίνακα.

Κατ' αυτόν τον τρόπο είναι δυνατόν να χρησιμοποιείται κάθε φορά η πιο αποτελεσματική υλοποίηση του ΑΤΔ στοίβα, ανάλογα με την εφαρμογή, με ελάχιστες (κανονικά καθόλου) τροποποιήσεις του κύριου προγράμματος.

Στο σημείο αυτό παρατηρεί κανείς έντονα το πλεονέκτημα της απόκρυψης της πληροφορίας που επιτυγχάνεται ακολουθώντας τη μέθοδο του ΑΤΔ και ειδικότερα της Ολικής Απόκρυψης.

Υλοποίηση του ΑΤΔ ουρά με συνδεδεμένη λίστα

Η υλοποίηση του ΑΤΔ ουρά με δείκτες είναι όμοια με εκείνη της στοίβας με τη μόνη διαφορά ότι εδώ χρειάζονται δύο δείκτες οι οποίοι δείχνουν το εμπρός και το πίσω μέρος της ουράς. Στην υλοποίηση της ουράς με συνδεδεμένη λίστα ο δείκτης embros δείχνει τον πρώτο κόμβο ενώ ο δείκτης piso τον τελευταίο κόμβο.



Νέα Τεχνική Υλοποίησης

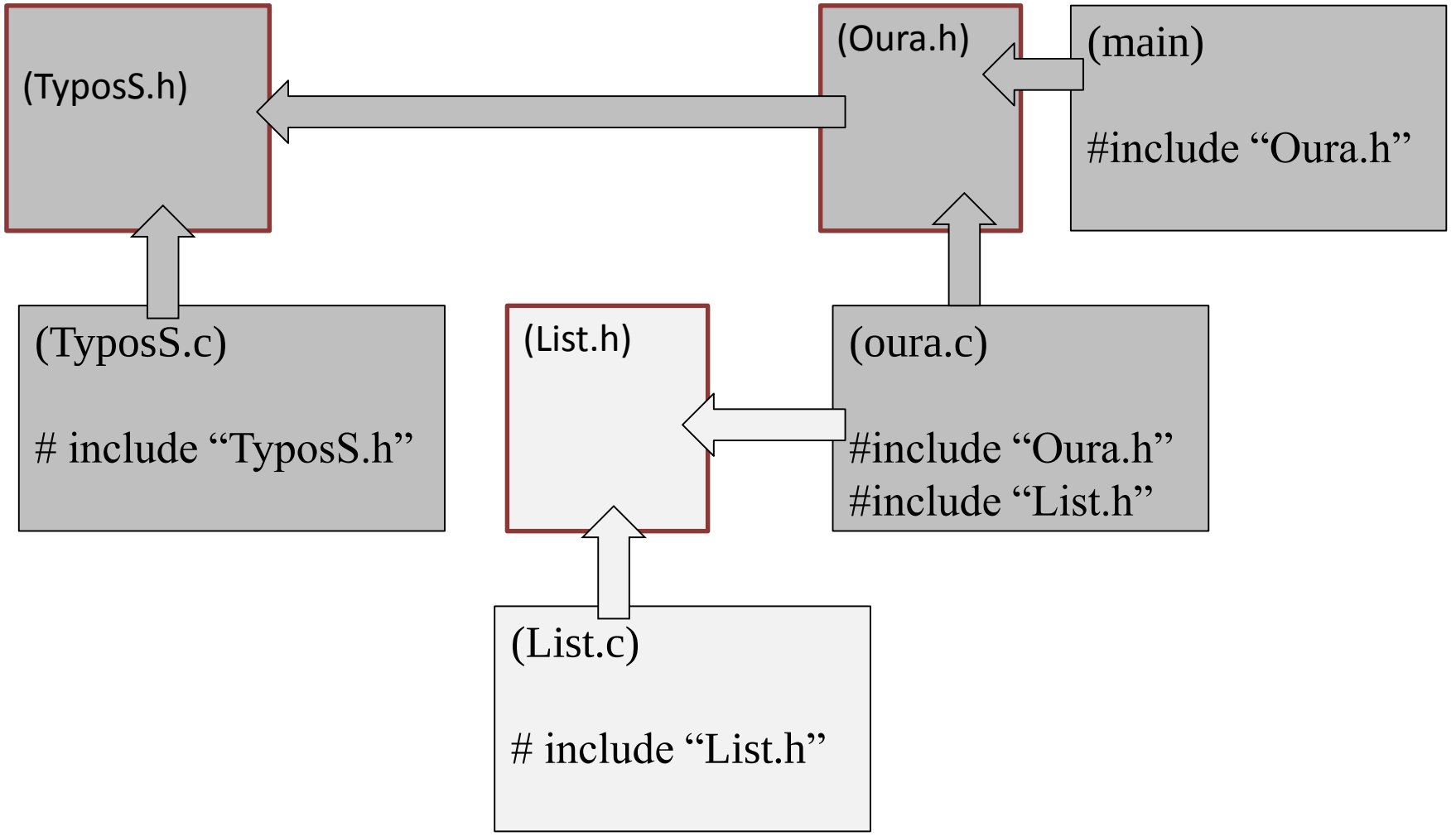
- Θα υλοποιήσουμε την Ουρά με συνδεδεμένη λίστα, αλλά επιπλέον θα χρησιμοποιήσουμε τις πράξεις του ΑΤΔ Λίστα για να υλοποιήσουμε πράξεις του ΑΤΔ Ουρά.
- Υλοποιημένη πλήρως στο 7.9 των σημειώσεων (+ κώδικας)

Το oyra.h (Ολική Απόκρυψη)

```
typedef struct QueueStruct *HandleOyras;

HandleOyras Oyra_dimiourgia();
void Oyra_katastrofi (HandleOyras *oura);
int Oyra_keni (HandleOyras oura);

void Oyra_prosthesi(HandleOyras *oura,
      TStoixeioyOyras stoixeio, int *yperxeilisi);

void Oyra_apomakrynsh(HandleOyras *oura,
      TStoixeioyOyras *stoixeio, int *ypoxeilisi);
```

Χρήση της διεπαφής της Λίστας στο .c

```
#include “Oyra.h”
#include "List_pointer.h”
/* συμπεριλαμβάνουμε την διεπαφή List_pointer.h */

typedef struct QueueStruct
{ info_deikti info_list;
} QueueStruct;          /*o typos ths oyras – lista*/
```


(TyposS.h)
(Oura.h)
(main)
#include “Oura.h”
(TyposS.c)
include “TyposS.h”
(List.h)
(oura.c)
#include “Oura.h”
#include “List.h”
(List.c)
include “List.h”

```
HandleOyras Oyra_dimiourgia(){
        HandleOyras ThisQueue;
        ThisQueue = malloc(sizeof(QueueStruct));

        LIST_dimiourgia(&ThisQueue->info_list);
                                    /* μέσω ΑΤΔ List */

        return ThisQueue;
}
```

Παρατηρούμε ότι η δημιουργία της Λίστας γίνεται μέσω της LIST_dimiourgia. Σε αυτό το επίπεδο αφαίρεσης δεν ξέρουμε ποια υλοποίηση λίστας έχουμε (μονή, διπλή, με κεφαλή, κλπ). Για να το δούμε πρέπει να δούμε την υλοποίηση της Λίστας (στο .c της Λίστας) .

Επιλέγουμε με δείκτη στο τέλος για καλύτερη αποδοση

```
int Oyra_keni(HandleOyras oura){
      return LIST_keni(oura->info_list);
}
```

```
void Oyra_katastrofi(HandleOyras *oura){
      LIST_katastrofi(&(*oura)->info_list);
      free(*oura);
      *oura=NULL;
}
```

Όφελος από την χρήση του ΑΤΔ Λίστα: Η καταστροφή (free) όλων των κόμβων γίνεται μέσω της LIST_katastrofi. Κάνουμε λίγη δουλειά.

Προσέξτε τα ονόματα των πράξεων: Βάζουμε πάντα το πρόθεμα του ΑΤΔ για να ξεχωρίσουν τα ονόματα σε διαφορετικούς ΑΤΔ (Oyra_keni, LIST_keni)

```
void Oyra_prosthesi(HandleOyras *oura,
      TStoixeioyOyras stoixeio, int *yperxeilisi)
{
  int error=0;
  LIST_eisagogi(&((*oura)->info_list), stoixeio,
                                  yperxeilisi);
}
```

Η List_eisagogi εισάγει στην λίστα με δείκτη embros, το stoixeio, μετά τον προδείκτη piso. Αν piso == NULL η εισαγωγή γίνεται στην αρχή.

```
void Oyra_apomakrynsh (HandleOyras *oura,
      TStoixeioyOyras *stoixeio, int *ypoxeilisi)
{
  if (Oyra_keni(*oura))
      *ypoxeilisi=1;
  else
  {  *ypoxeilisi=0;
     LIST_periexomeno((*oura)->info_list->arxi,
                                          stoixeio);
     LIST_diagrafi_arxi(&((*oura)->info_list),
                                          ypoxeilisi);

  }
}
```

Υλοποίηση μέσω άλλου ΑΤΔ

Πλεονεκτήματα

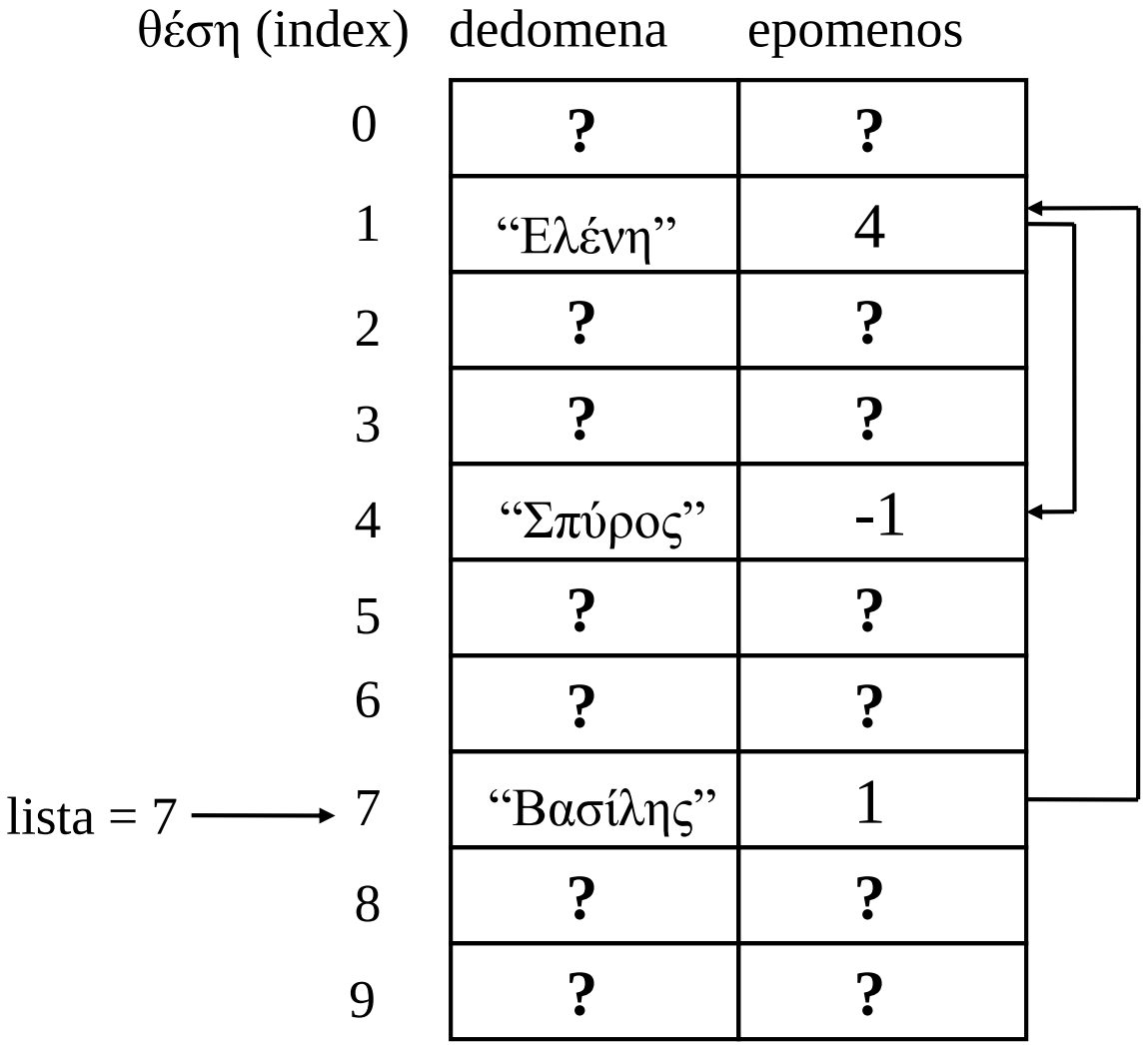
- Αναπτύσσουμε λιγότερο κώδικα (π.χ. καταστροφή)
- Επαναχρησιμοποιούμε δοκιμασμένο (;) κώδικα

Μειονεκτήματα

- Η διεπαφή να μην ταιριάζει ακριβώς (π.χ. Piso)
- Απόδοση εξαρτάται από (κρυμμένο) κώδικα

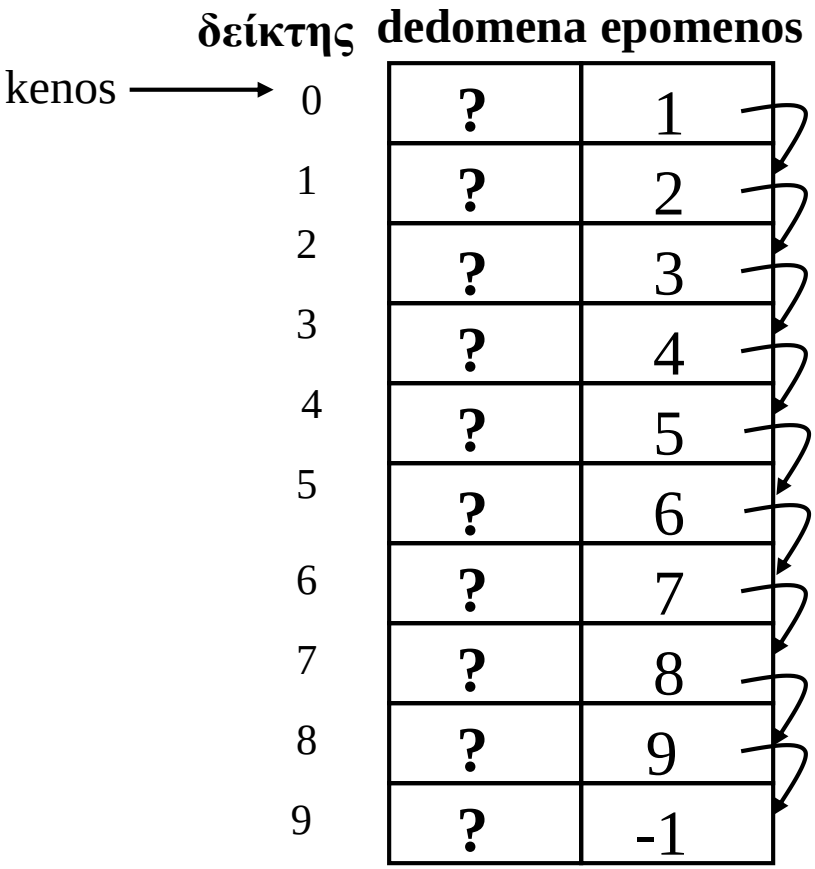
Σχεδιαστική Επιλογή
Συνδεδεμένη Δομή με Πίνακα

2η Υλοποίηση του ΑΤΔ συνδεδεμένη λίστα με πίνακα
Κόμβοι από δεδομένα και θέσεις πίνακα



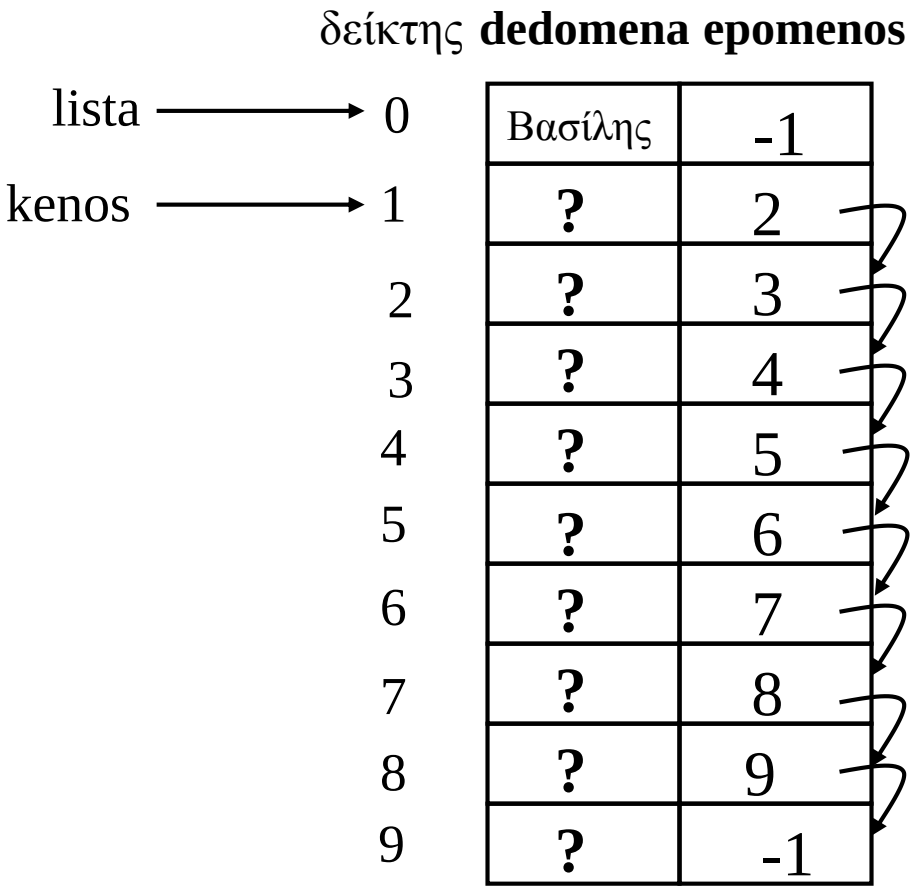
Στη συνέχεια θα πρέπει να γνωρίζουμε την οργάνωση του πίνακα αυτού προκειμένου να γίνει εισαγωγή ή διαγραφή κάποιου στοιχείου του.

Φυσικό είναι να γνωρίζουμε την αρχική κατάσταση του πίνακα όπως στο σχήμα :

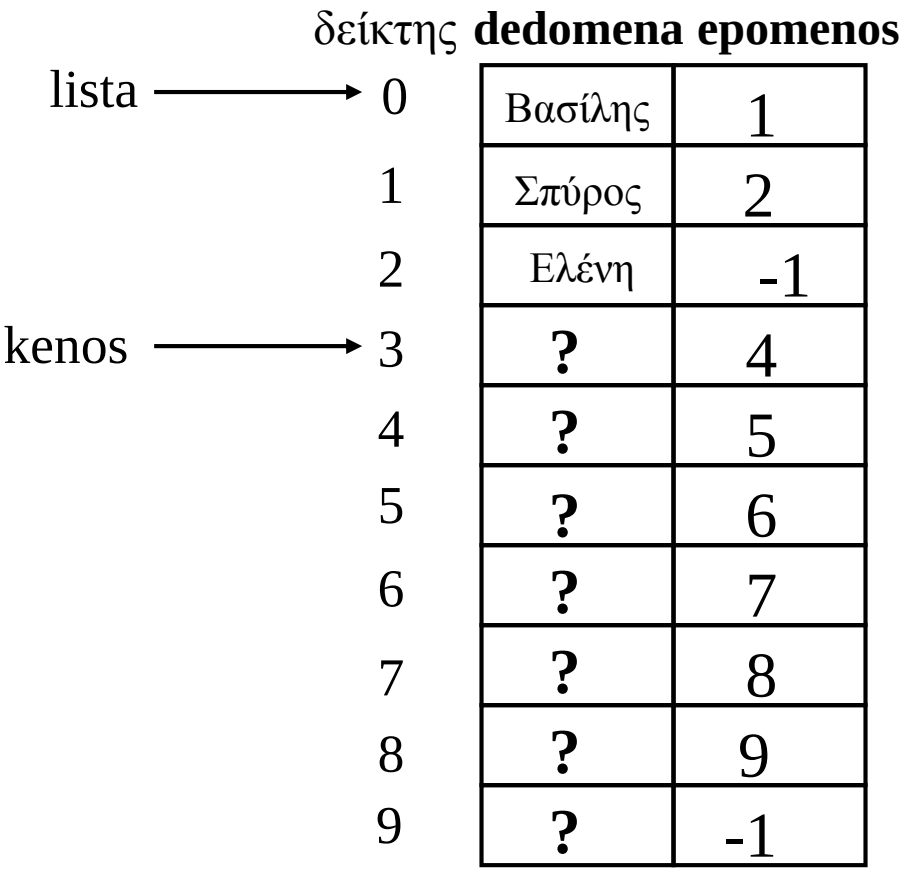


-1 συνθήκη τέλους (έξω από όριο πίνακα (null)

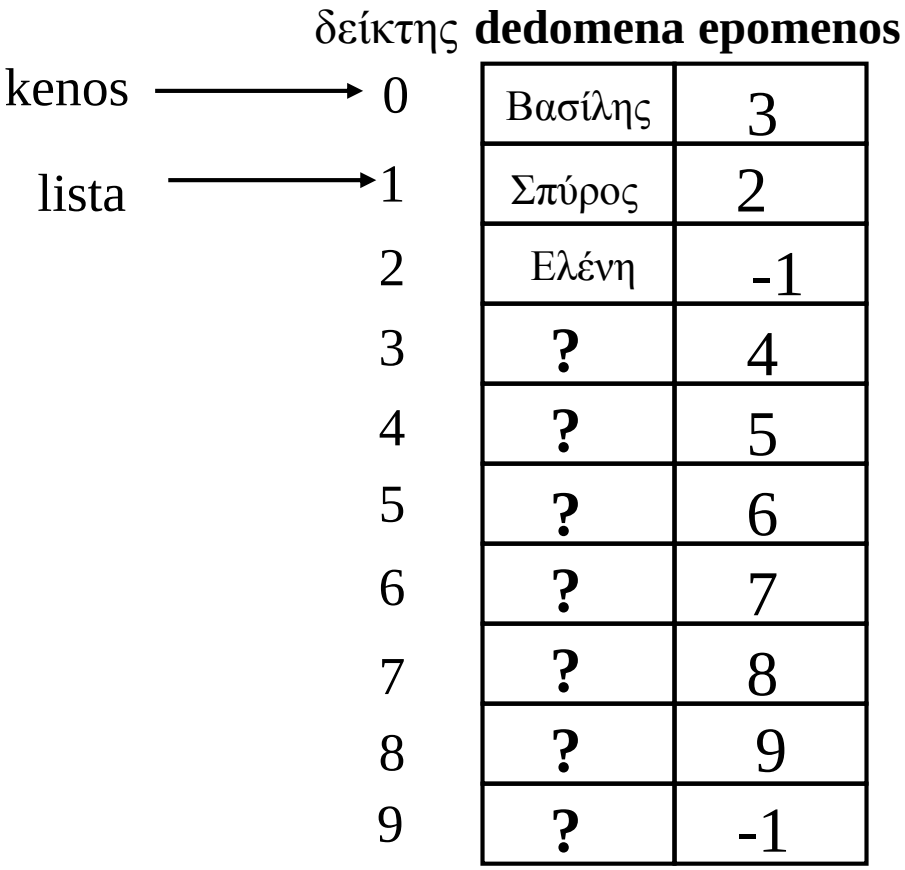
Στη συνέχεια, ας υποθέσουμε ότι ‘Βασίλης’ είναι το πρώτο όνομα που θα εισαχθεί σε μια συνδεδεμένη λίστα. Το όνομα αυτό θα τοποθετηθεί στην πρώτη θέση του πίνακα του και θα έχουμε το Σχήμα :



Όταν διαγράφεται ένας κόμβος από τη συνδεδεμένη λίστα πρέπει η θέση του να συμπεριληφθεί στις κενές θέσεις του πίνακα ώστε να ξαναχρησιμοποιηθεί αργότερα για την αποθήκευση κάποιου άλλου στοιχείου της λίστας. Για παράδειγμα, ας υποθέσουμε ότι έχουμε τη συνδεδεμένη λίστα του σχήματος



Η διαγραφή του ονόματος ‘Βασίλης’ έχει σαν αποτέλεσμα αυτό του σχήματος. Ας σημειωθεί ότι δεν είναι αναγκαία η φυσική απομάκρυνση του ονόματος ‘Βασίλης’ από το τμήμα των δεδομένων.



```
#define plithos ...
#define null -1          // για να μοιάζει με NULL
#define not_spec -1
typedef ... TStoixeioyListas;

typedef struct {
      TStoixeioyListas dedomena;
      int epomenos;
} typos_komvou;

typedef struct typos_listas{
      typos_komvou komvos[plithos];
      int HeadList; /*thesi tou prwtou xrhsimou kombou*/
      int HeadKenos;/*thesi tou prwtou eley8erou kombou*/
} typos_listas;
```

```
void arxiki_katastasi(typos_listas * ListPtr){
     int i;
/* σύνδεση των κενών κόμβων μεταξύ τους */
     for ( i=0; i<plithos-1; i++ )
          ListPtr->komvos[i].epomenos = i+1;

/* ένδειξη τέλους της λίστας */
     ListPtr->komvos[plithos-1].epomenos=null;
     ListPtr->HeadKenos = 0;
     list->HeadList=not_spec;
}
```

Για την εκτέλεση της παραπάνω διαδικασίας απαιτείται πρώτα ο εντοπισμός της κενής θέσης όπου θα εισαχθεί ο νέος κόμβος.

Η pare_komvo(p) τοποθετεί στο p την τιμή της kenos, δηλαδή την θέση του πίνακα komvos που είναι κενή και πρόκειται να τοποθετηθεί ο νέος κόμβος. (κατ' αναλογία της malloc ())

Ταυτόχρονα, διαγράφει (αφαιρεί) τη θέση αυτή από τη λίστα των κενών θέσεων. (κατ' αναλογία της διαχείρισης του σωρού).

```
void pare_komvo(typos_listas *list, int *p, int *error)
{ error =0;
  *p = list->HeadKenos;
  if (list->HeadKenos!=not_spec)
      list->HeadKenos=
            list->komvos[list->HeadKenos].epomenos;
  else
      *error=1;
}
```

Η διαδικασία που υλοποιεί μόνο την αποδέσμευση του χώρου του κόμβου που διαγράφεται είναι η ακόλουθη:

```
void apodesmeysi(typos_listas *list, int p)
{
      list->komvos[p].epomenos = list->HeadKenos;
      list->HeadKenos = p;
}
```

Κατ' αναλογία **του `free()`**

Η διαδικασία για τη δημιουργία μιας κενής συνδεδεμένης λίστας τοποθετεί την τιμή null (-1) στο δείκτη lista και αρχικοποιεί τον πίνακα με όλα τα στοιχεία στην λίστα των κενών.

```
void dimiourgia(typos_list *listPtr){
  arxiki_katastasi(listPtr); /* οι κενοί κόμβοι */
}
```

Για τον έλεγχο αν μια συνδεδεμένη λίστα είναι κενή έχουμε το ακόλουθο υποπρόγραμμα :

```
int LIST_keni(typos_listas lista)
{/*Pro: Dhmioyrgia listas
  *Meta:epistrefei 1 an h lista einai kenh,
          diaforetika 0 */

     return (lista.HeadList == not_spec );
}
```

```
void LIST_eisagogi(typos_listas *list,
                   TStoixeioyListas stoixeio,
                   int prodeiktis, int *error){
/* Pro: Dhmioyrgia listas
   Meta: An o prodeiktis einai null tote o kombos me ta
      dedomena stoixeio exei eisax8ei sthn arxh ths
      listas (symvash) alliws eisagetai meta ton kombo
      poy deixnei o prodeiktis */

      *error=0;
      if (prodeiktis == not_spec) /*eisagwgh sthn arxh */
        eisagogi_arxi(list, stoixeio, error);
      else
        eisagogi_meta(list, prodeiktis, stoixeio, error);
}
```

```
void eisagogi_arxi(typos_listas *list,
                   TStoixeioyListas stoixeio,int *error)
{
  int prosorinos;

  pare_komvo(list,&prosorinos,error);
  TSlist_setValue(&(list->komvos[prosorinos].dedomena),
                                          stoixeio);

  list->komvos[prosorinos].epomenos=list->HeadList;
  list->HeadList=prosorinos;
}
```

```
void eisagogi_meta(typos_listas *list, int prodeiktis,
            TStoixeioyListas stoixeio, int *error){

  int prosorinos; /*Deixnei to neo kombo poy 8a
eisax8ei*/

  pare_komvo(list, &prosorinos, error);

  TSlist_setValues(
      &(list->komvos[prosorinos].dedomena), stoixeio);

  list->komvos[prosorinos].epomenos=
                  list->komvos[prodeiktis].epomenos;

  list->komvos[prodeiktis].epomenos=prosorinos;
}
```

```
void LIST_diagrafi(typos_listas *lista, int prodeiktis,
                   int *error)
{ *error=0;
  if (LIST_keni(*lista))
       *error=1;
  else
  {  int prosorinos;/* o kombos poy 8a diagrafei*/

     if (prodeiktis != null)      /* όχι στην αρχή */
     {   prosorinos = lista->komvos[prodeiktis].epomenos;
         lista->komvos[prodeiktis].epomenos =
                   lista-> komvos[prosorinos].epomenos;
     }
     else if (prodeiktis == null) /* στην αρχή */
     {  prosorinos = lista -> HeadList;
         lista->HeadList = lista->komvos[HeadList].epomenos;
     }
     apodesmeysi(lista, prosorinos);
}
```

Τέλος Ενότητας

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Σημειώματα

Σημείωμα Αναφοράς

Σημείωμα Αδειοδότησης

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.