Λίστες (Lists)

Ο ΑΤΔ λίστα είναι μια συλλογή στοιχείων του ίδιου τύπου με γραμμική δομή.

Βασικές Πράξεις

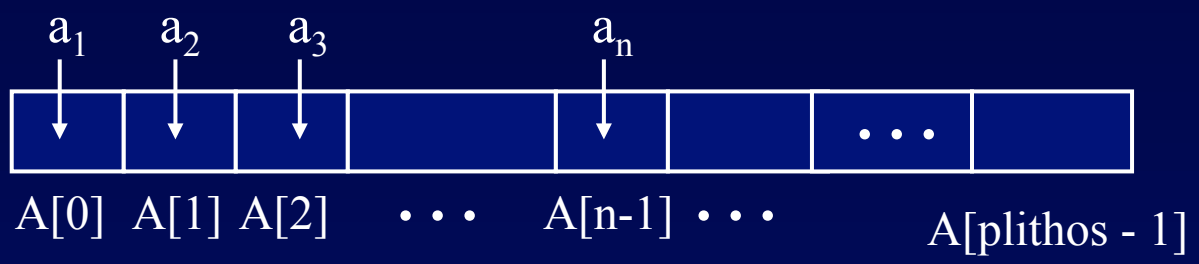
- Δημιουργία
- Κενή
- Περιεχόμενο
- Προχώρησε
- Εισαγωγή Μετά
- Εισαγωγή Πριν
- Διαγραφή
- Αναζήτηση
- Μήκος

Ο ΑΤΔ Ακολουθιακή Λίστα

Στον ΑΤΔ ακολουθιακή λίστα η φυσική διάταξη των στοιχείων της (στην υλοποίηση) ταυτίζεται με εκείνη της λογικής τους διάταξης.

Υλοποίηση με πίνακα

Η **ακολουθιακή υλοποίηση** έχει σαν βασικό χαρακτηριστικό ότι διαδοχικά στοιχεία της λίστας αποθηκεύονται σε διαδοχικές θέσεις του πίνακα. Ετσι 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., όπως δείχνει το ακόλουθο σχήμα :



Εκτός από την αποθήκευση των στοιχείων της λίστας στον πίνακα είναι, αποθηκεύουμε σε μια μεταβλητή και το τρέχον πλήθος των στοιχείων που περιέχει η λίστα. Έχουμε λοιπόν τις δηλώσεις :

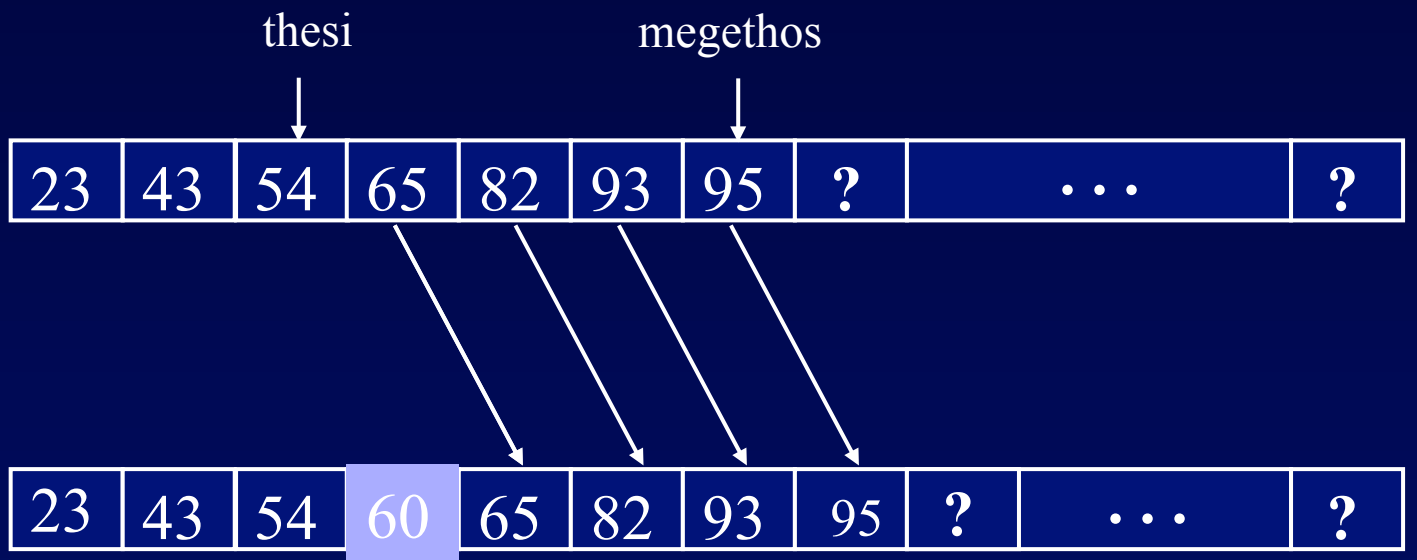
```
#define plithos ...
typedef ... typos_stoixeiou;
typedef int typos_deikti;
typedef typos_stoixeiou     typos_pinaka[plithos];
typedef struct {
                        typos_pinaka pinakas;
                        typos_deikti megethos;
                } typos_listas;

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,  typos_deikti thesi)
/*Προ : Δημιουργία λίστας.
   Μετά: Επιστρέφει την επόμενη θέση της thesi αλλιώς
   επιστρέφει το -1. */
{
        if (( thesi < 0 ) || ( thesi > = lista.megethos – 1 ))
        {
                printf(" Ο δείκτης thesi είναι εκτός διαστήματος");
                return  (-1);
         }
        else
                return ( thesi++ );
}
```



Για την εισαγωγή ενός νέου στοιχείου στη λίστα θα πρέπει να δημιουργηθεί χώρος. Η υλοποίηση της διαδικασίας της εισαγωγής ενός νέου στοιχείου στη λίστα παρουσιάζεται στη συνέχεια.

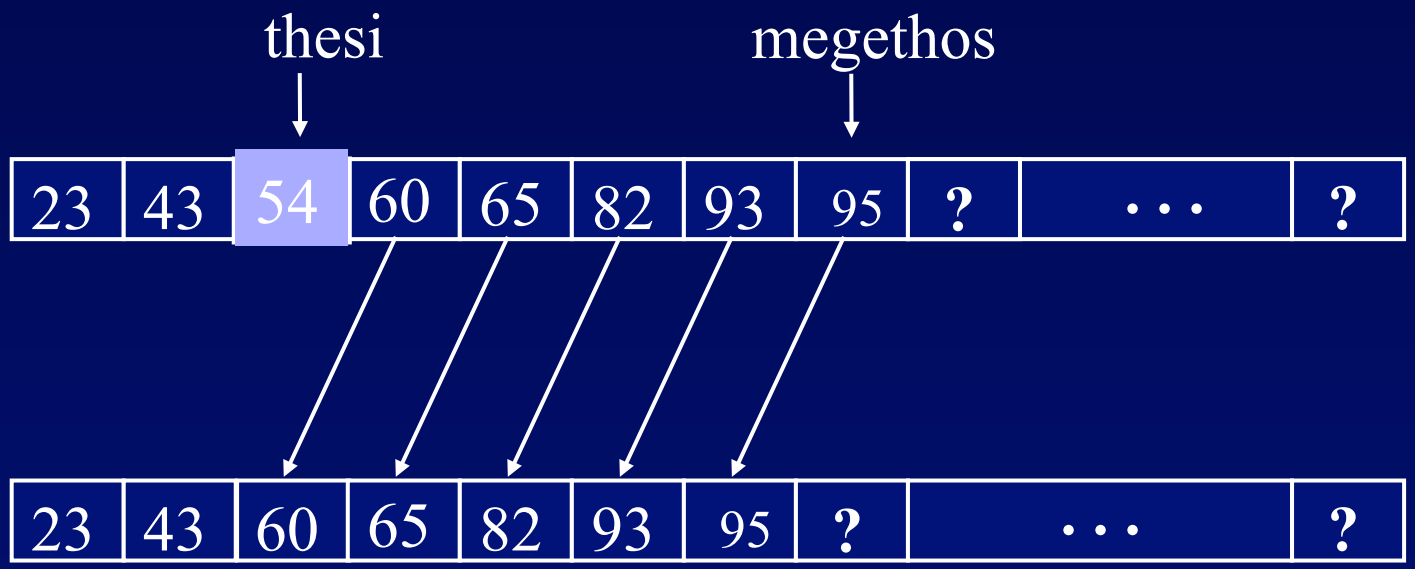
Πολυπλοκότητα :O(n)

```
void eisagogi_meta( typos_listas *lista, typos_stoixeiou stoixeio,
                                                typos_deikti thesi)
/*   Προ : Δημιουργία της λίστας.
     Μετά:  Εισάγεται το στοιχείο stoixeio στη λίστα μετά το
     στοιχείο που  βρίσκεται στη θέση thesi */
{
      int i;
      if (lista->megethos == plithos)
            printf(" Η λίστα είναι  γεμάτη");
      else
            if (thesi < 0 )  or  (thesi > lista -> megethos - 1)
                  printf(" Ο δείκτης thesi είναι εκτός διαστήματος");
            else
```

συνέχεια

```
    {
        /* Μετακίνηση στοιχείων μια θέση δεξιά */
        for (i = lista->megethos-1 ; i >= thesi+1 ; i--)
            lista->pinakas[i+1] = lista->pinakas[i];

        /* εισαγωγή του νέου στοιχείου μετά τη θέση
           thesi και αύξηση του μεγέθους της λίστας */

        lista->pinakas[thesi+1]=stoixeio;
           lista->megethos++;
    }
}
```

Διαγραφή



```
void diagrafi(typos_listas *lista, typos_deikti thesi)
/* Προ : Δημιουργία της λίστας.
    Μετά: Διαγράφεται το στοιχείο της *lista που βρίσκεται στη
θέση thesi.
*/

{
        int i;
        if (keni(*lista))
                printf("Η λίστα είναι κενή");
        else
```

συνέχεια

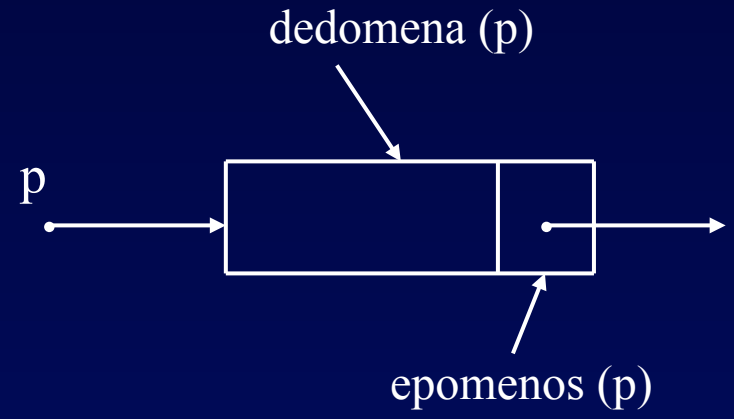
```
        }
            if ((thesi<0) || (thesi > lista->megethos -1))
                    printf(" Η thesi είναι εκτός διαστήματος");

            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];
           }
        }
}
```

Ο ΑΤΔ Συνδεδεμένη Λίστα

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. Επίσης υπάρχει ένας εξωτερικός δείκτης lista, ο οποίος περιέχει τη διεύθυνση του πρώτου κόμβου της λίστας.

Σχηματική Παράσταση Συνδεδεμένης Λίστας

- Τα βέλη συμβολίζουν δείκτες
- Η τελεία στο τμήμα επόμενο του τελευταίου κόμβου αντιπροσωπεύει το μηδενικό δείκτη (null)

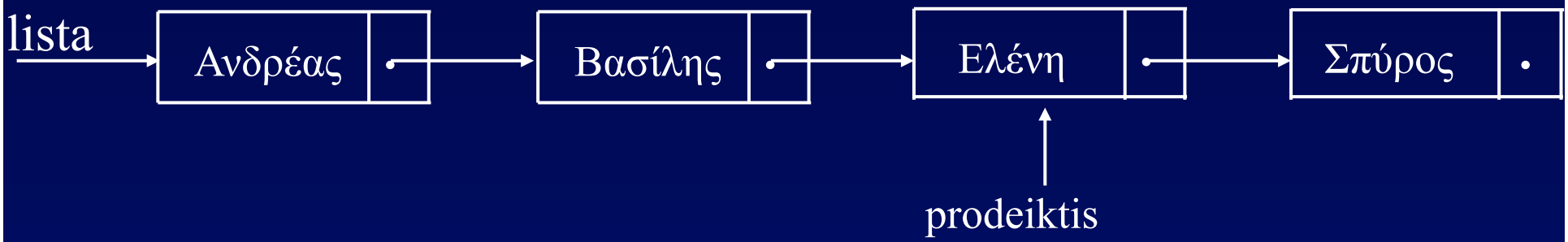


apodesmeysi (p)

pare_komvo (p)

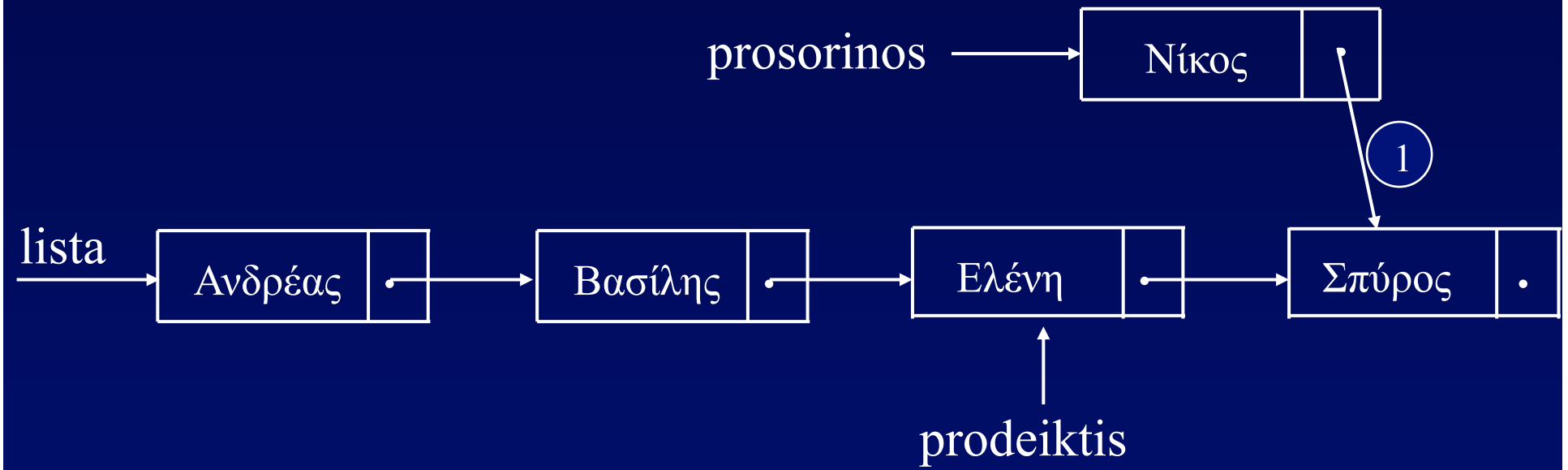
Εισαγωγή

Εισαγωγή μετά από έναν κόμβο :

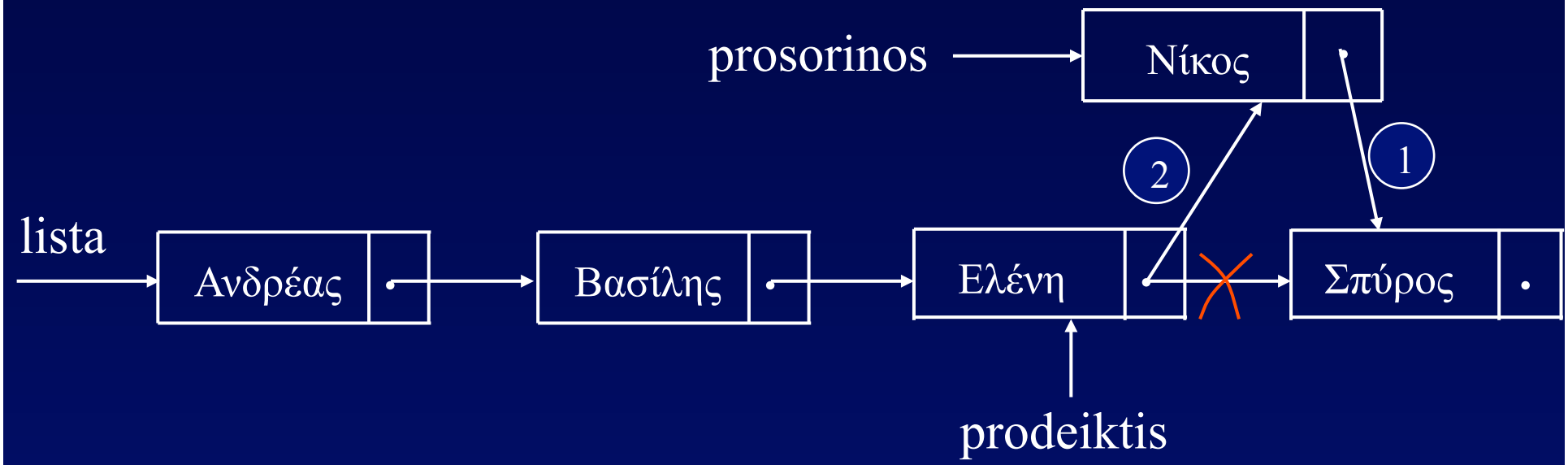


pare_komvo (prosorinos)

dedomena (prosorinos) = “Νίκος”



epomenos (prosorinos) = epomenos (prodeiktis) (1)

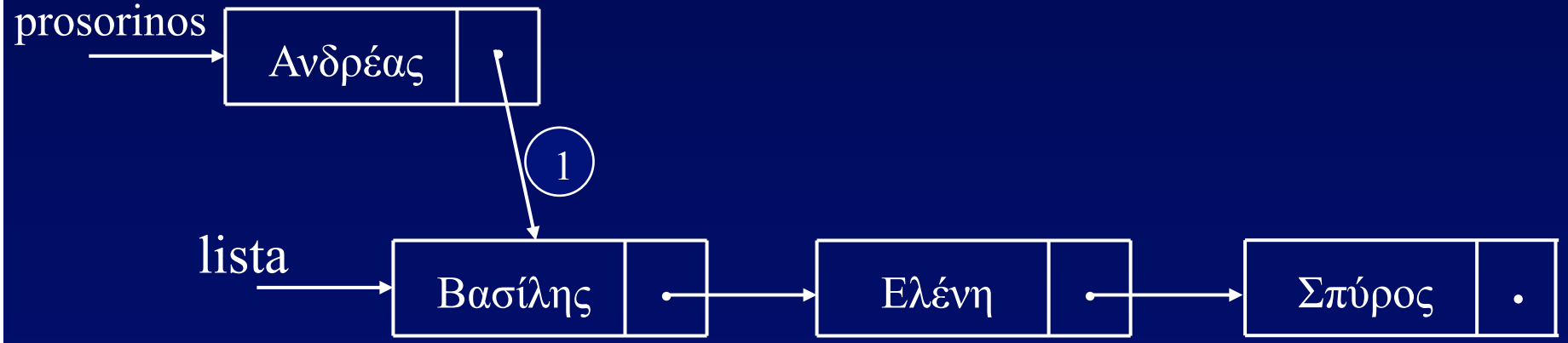


epomenos (prodeiktis) = prosorinos (2)

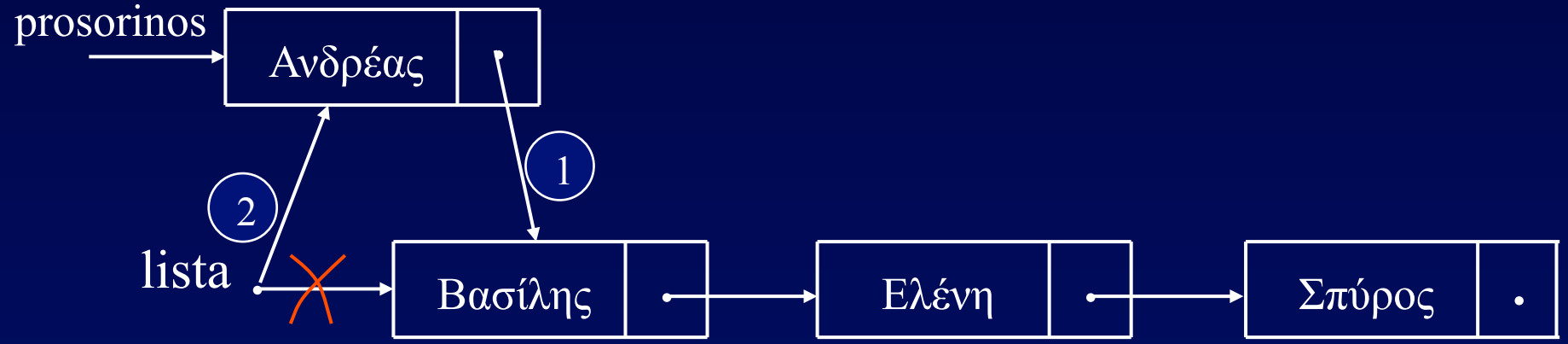
Εισαγωγή στην αρχή :

pare_komvo(prosorinos)

dedomena(prosorinos) = “Ανδρέας”



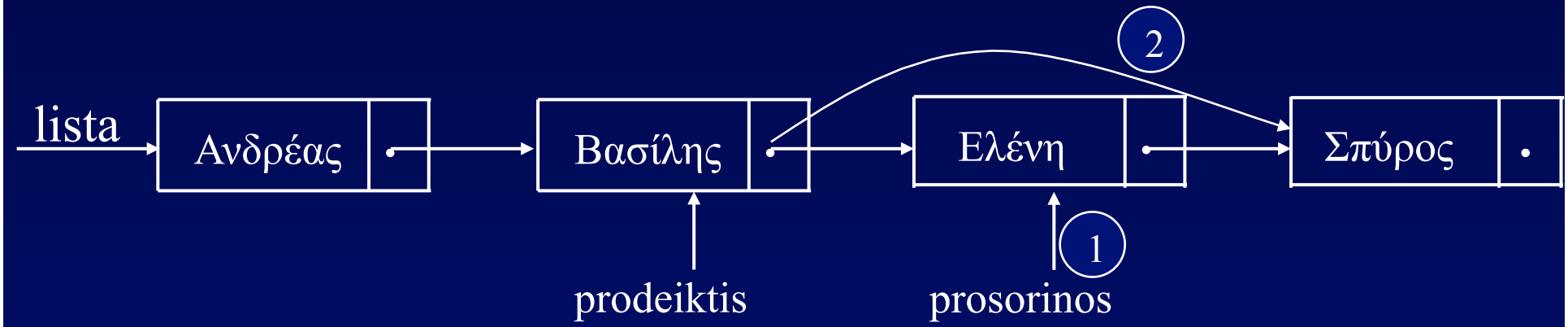
epomenos(prosorinos) = lista (1)



lista = prosorinos (2)

Διαγραφή

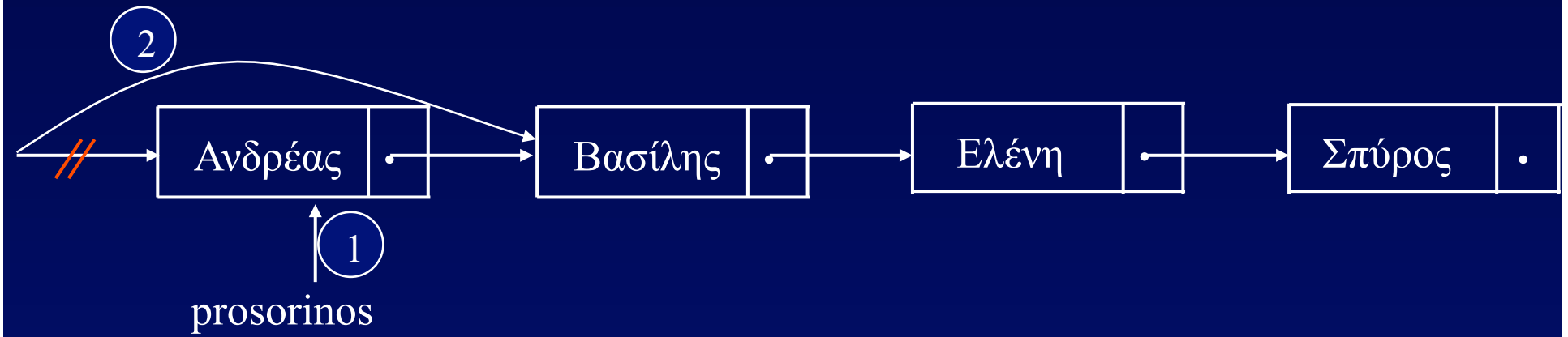
Διαγραφή μετά από έναν κόμβο :



prosorinos = epomenos (prodeiktis) (1)

epomenos (prodeiktis) = epomenos (prosorinos) (2)

Διαγραφή στην αρχή :



prosorinos = lista (1)

lista = epomenos (prosorinos) (2)

Τόσο για τον υπολογισμό του **μήκους** όσο και για το πρόβλημα της **αναζήτησης** είναι αναγκαία η επίσκεψη όλων των κόμβων μιας λίστας ξεκινώντας από τον πρώτο. Η πράξη αυτή αποτελείται από τα εξής βήματα:

1. Χρησιμοποιείται ένας βοηθητικός δείκτης που διατρέχει όλους τους κόμβους της λίστας. Ο δείκτης αυτός είναι ο trexon και αρχικά δείχνει τον πρώτο κόμβο.

2. Επεξεργασία πεδίου dedomena(trexon).

3. Μετακίνηση του δείκτη trexon στον επόμενο κόμβο.

Ο αλγόριθμος της επίσκεψης των κόμβων μιας συνδεδεμένης λίστας είναι ο ακόλουθος :

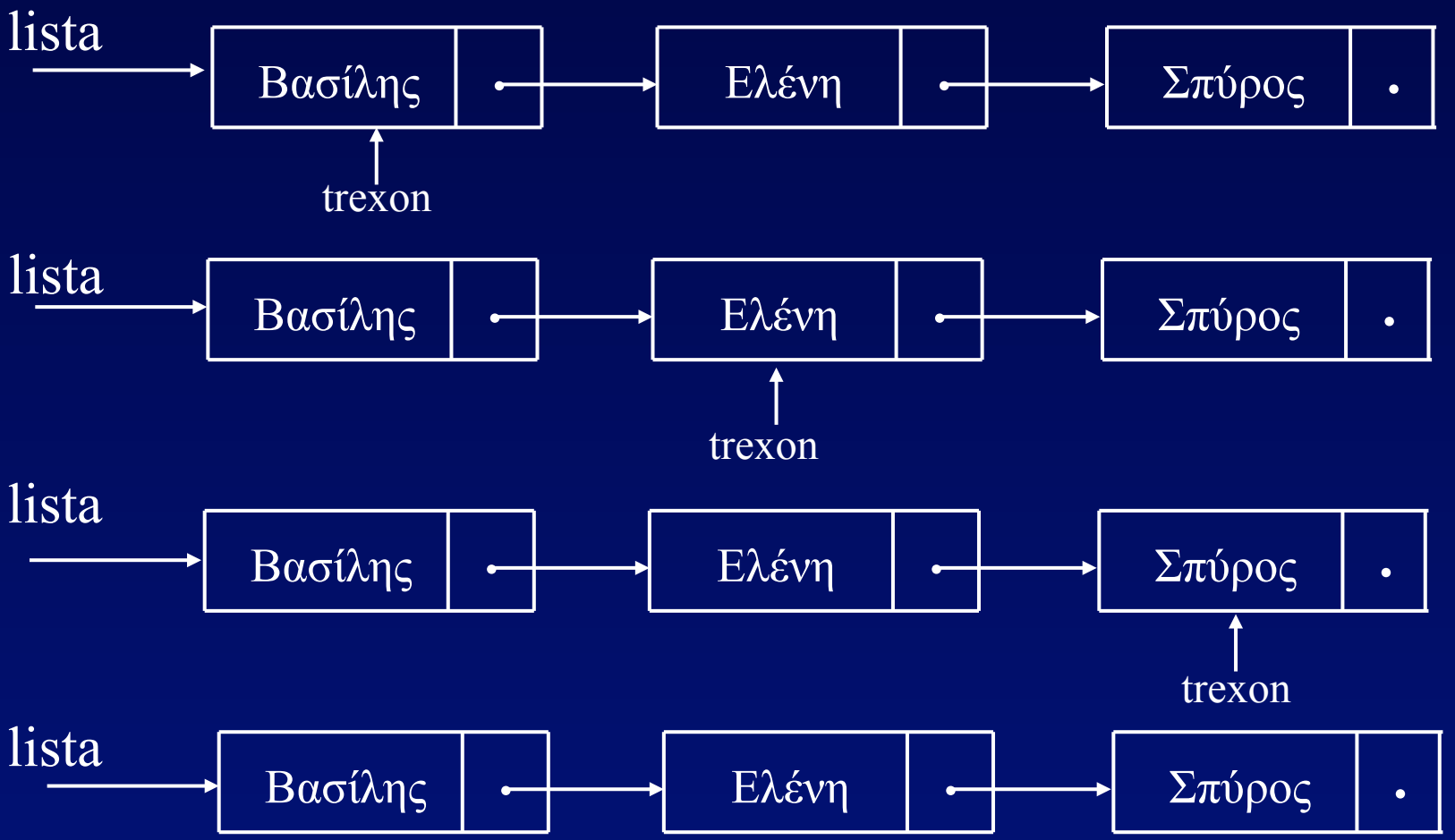
1. trexon = lista.

2. Οσο trexon != NULL να εκτελούνται :

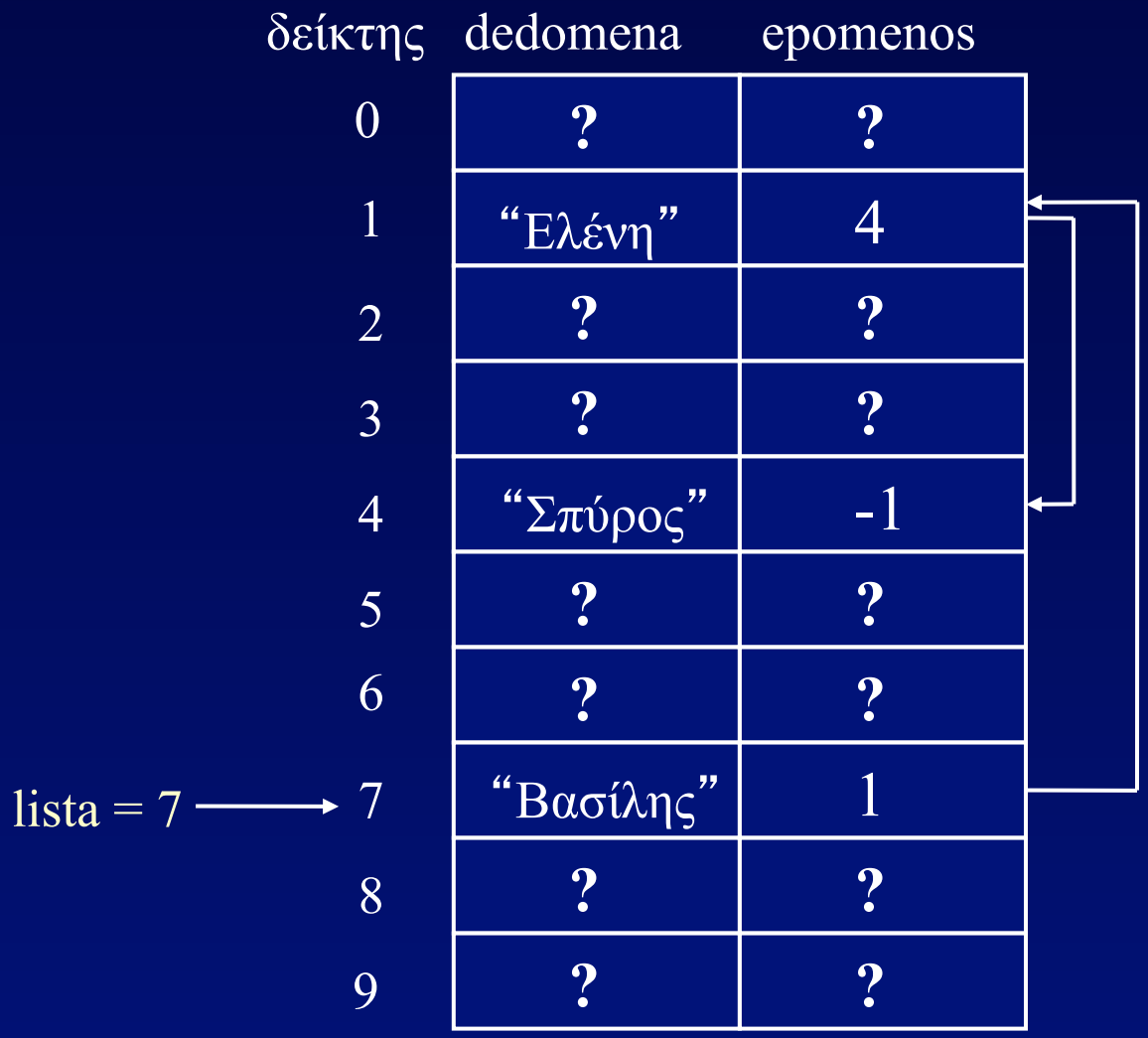
α. Επεξεργασία του dedomena(trexon).

β. trexon = epomenos(trexon).

Η εφαρμογή του αλγόριθμου για την λίστα των ονομάτων εμφανίζεται στο ακόλουθο σχήμα :

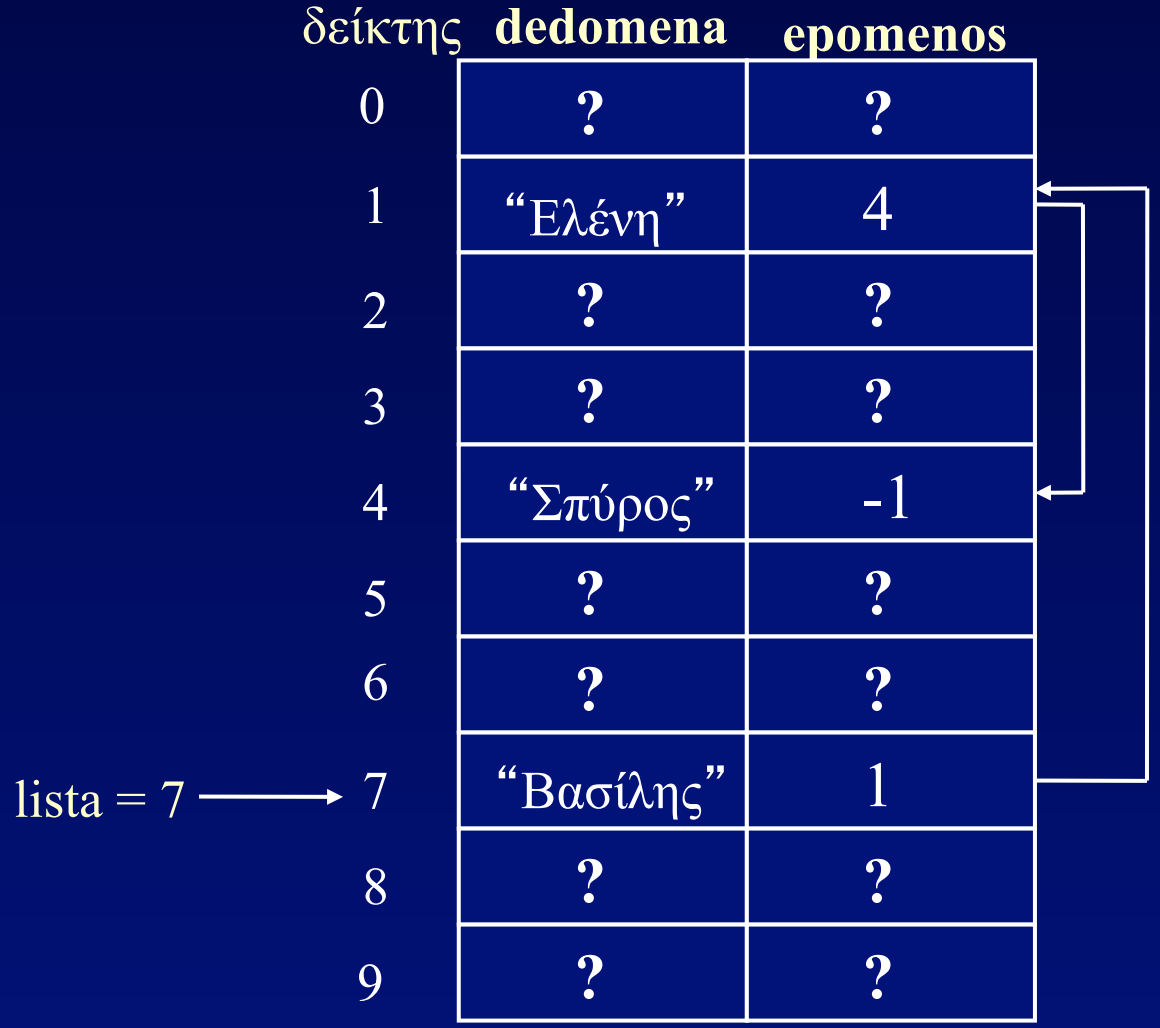


Υλοποίηση του ΑΤΔ συνδεδεμένη λίστα με πίνακα



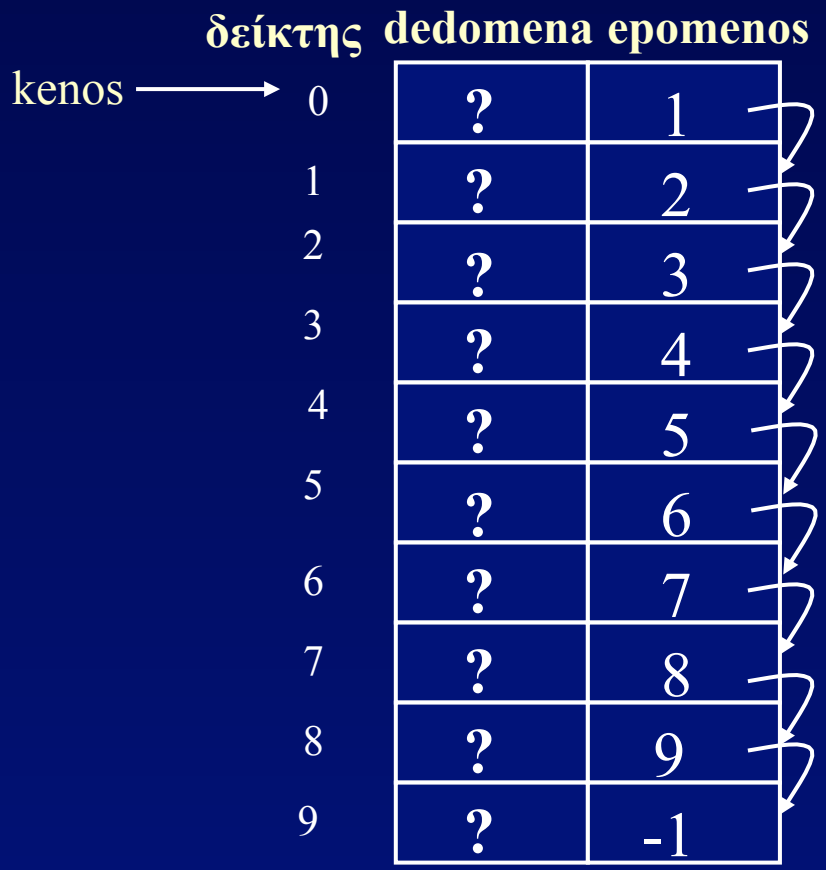
```
#define plithos ...
#define null -1
typedef ... typos_stoixeiou;
typedef int typos_deikti;
typedef struct
{
        typos_stoixeiou dedomena;
        typos_deikti epomenos;
} typos_komvou;

typedef typos_komvou      pinakas_komvon[plithos];

pinakas_komvon   komvos;
typos_deikti   kenos,  lista;
```

Η αποθήκευση της λίστας των ονομάτων παρουσιάζεται στο ακόλουθο σχήμα όπου παρατηρούμε ότι η τοποθέτηση των τριών εγγραφών μπορεί να γίνει οπουδήποτε μέσα στον πίνακα με 10 στοιχεία.



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

Δημιουργούμε την εξής αρχική κατάσταση στον πίνακα:



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

        /* ένδειξη τέλους της λίστας */
        komvos[plithos-1].epomenos = null;
        kenos = 0;
}
```

Παρατήρηση : Η komvos και kenos είναι καθολικές μεταβλητές !!

Στη συνέχεια, ας υποθέσουμε ότι ‘Βασίλης’ είναι το πρώτο όνομα που θα εισαχθεί σε μια συνδεδεμένη λίστα. Το όνομα αυτό θα τοποθετηθεί στην πρώτη διαθέσιμη θέση του πίνακα στην οποία δείχνει η μεταβλητή kenos:

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 δείκτη p την τιμή της kenos, η οποία αντιστοιχεί στην πρώτη διαθέσιμη θέση του πίνακα komvos που είναι κενή και πρόκειται να τοποθετηθεί ο νέος κόμβος. Ταυτόχρονα, διαγράφει τη θέση αυτή από τη λίστα των κενών θέσεων.

```
void pare_komvo(typos_deikti *p)
/*Προ : Καμμία.
  Μετά: Αν υπάρχει κάποιος διαθέσιμος κόμβος τότε ο
  δείκτης *p δείχνει έναν κενό κόμβο αλλιώς ο *p είναι
  null(-1). */
{
        *p = kenos;
        if (kenos != null)
                kenos = komvos[kenos].epomenos;
}
```

Οταν 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

Τότε η διαγραφή του ονόματος ‘Βασίλης’ έχει σαν αποτέλεσμα αυτό του σχήματος.

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

```
void apodesmeysi(typos_deikti p)
{
        komvos[p].epomenos = kenos;
        kenos = p;
}
```

Η διαδικασία για τη δημιουργία μιας κενής συνδεδεμένης λίστας τοποθετεί την τιμή null στο δείκτη lista:

```
void dimiourgia(typos_deikti *lista)
{
        *lista = null;
}
```

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

```
int keni(typos_deikti lista)
{
        return (lista==null);
}
```

Η πράξη proxorise υλοποιείται από το ακόλουθο υποπρόγραμμα ::

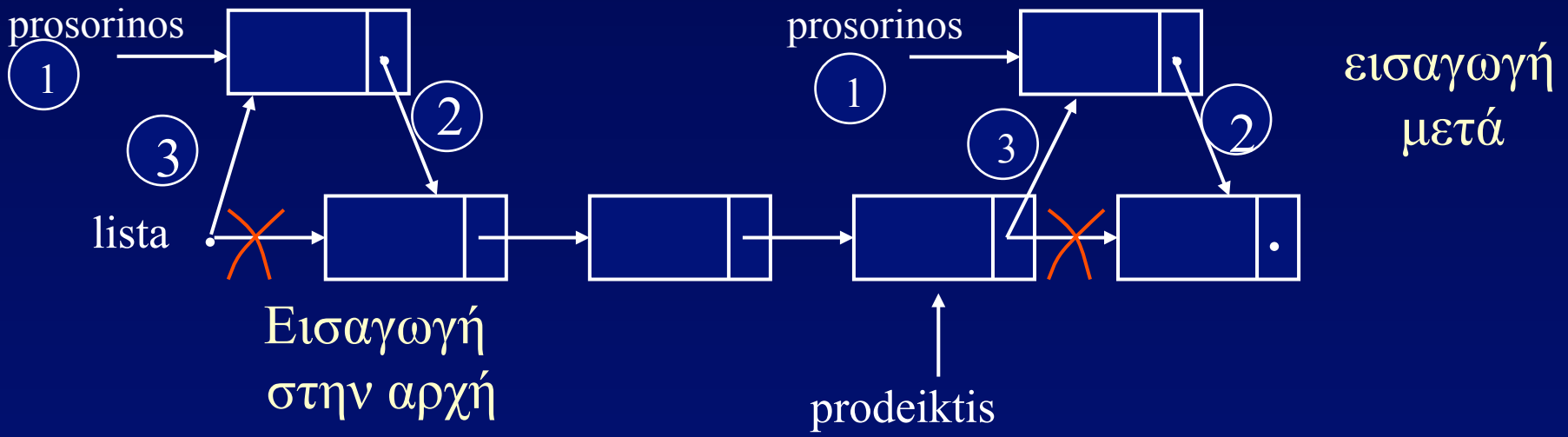
```
void proxorise(typos_deikti *p)
{
        *p = komvos[*p].epomenos;

}
```

Η πράξη της προσπέλασης στα δεδομένα ενός κόμβου (περιεχόμενο) υλοποιείται με το ακόλουθο υποπρόγραμμα:

```
typos_stoixeiou periexomeno(typos_deikti p)
/* Προ : Ο δείκτης p δείχνει ένα κόμβο με ορισμένο το πεδίο
         dedomena.
   Μετα  : Επιστρέφει τα δεδομένα του κόμβου που δείχνει ο
           δείκτης p.
*/
{
        return (komvos[p].dedomena);
}
```

Για τις διαδικασίες της εισαγωγής και διαγραφής θα υποθέσουμε καταρχήν ότι γνωρίζουμε τη θέση στην οποία πρόκειται να εισαχθεί ή να διαγραφεί ένας κόμβος. Ετσι ο αλγόριθμος της εισαγωγής είναι ο παρακάτω:

Αλγόριθμος εισαγωγής σε συνδεδεμένη λίστα



1. pare_komvo(prosorinos)

2. dedomena(prosorinos) = stoixeio

3. Αν prodeiktis == null τότε {εισαγωγή στην αρχή}
   α. epomenos(prosorinos) = lista
   β. lista = prosorinos
   } εισαγωγή στην αρχή

   διαφορετικά
   α. epomenos(prosorinos) = epomenos(prodeiktis)
   β. epomenos(prodeiktis) = prosorinos
   } εισαγωγή μετά

Στη συνέχεια ακολουθεί η υλοποίηση των δύο αυτών πράξεων.

```
void eisagogi_meta(typos_deikti prodeiktis, typos_stoixeiou
stoixeio)
/* Προ : Πρέπει ο prodeiktis να είναι δείκτης που δείχνει ένα
κόμβο  στη λίστα.
     Μετά: Ο κόμβος με δεδομένα stoixeio έχει εισαχθεί μετά τον
κόμβο  που δείχνει ο δείκτης prodeiktis. */
{
  typos_deikti  prosorinos;

  pare_komvo(&prosorinos);
  komvos[prosorinos].dedomena = stoixeio;
  komvos[prosorinos].epomenos = komvos[prodeiktis].epomenos;
  komvos[prodeiktis].epomenos = prosorinos;
}
```

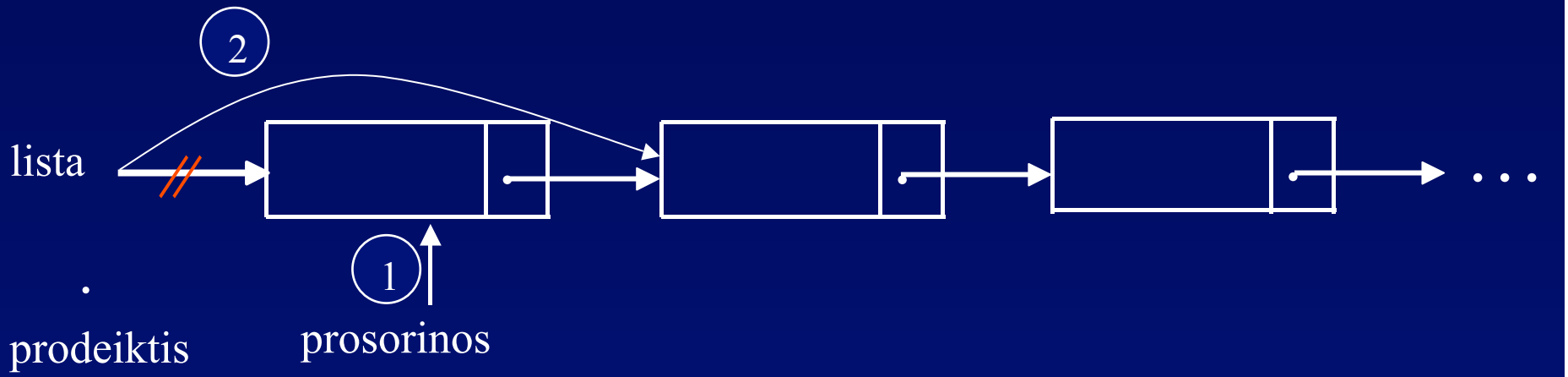
```
void eisagogi_prin(typos_deikti *p, typos_stoixeiou  stoixeio)
/* Προ : Πρέπει ο δείκτης *p να δείχνει τον κόμβο που είναι στην
αρχή της λίστας.
   Μετά: Ο κόμβος με δεδομένα stoixeio έχει εισαχθεί πριν τον
κόμβο  που έδειχνε ο δείκτης *p και ο *p  δείχνει πλέον τη νέα αρχή
της λίστας */
 {
        typos_deikti prosorinos;

        pare_komvo(&prosorinos);
        komvos[prosorinos].dedomena = stoixeio;
        komvos[prosorinos].epomenos = *p;
        *p = prosorinos;
}
```

Χρησιμοποιώντας τις ανωτέρω δύο πράξεις η εισαγωγή υλοποιείται ως εξής:

```
void eisagogi(typos_deikti *lista, typos_stoixeiou stoixeio,
                                        typos_deikti prodeiktis)
/* Προ : Πρέπει ο prodeiktis να είναι null(-1) ή  να δείχνει ένα κόμβο
      μέσα στη λίστα.
   Μετά: Αν ο prodeiktis είναι null(-1) τότε ο κόμβος με δεδομένα
      stoixeio έχει εισαχθεί στην αρχή της λίστας αλλιώς ο κόμβος με
      δεδομένα stoixeio έχει εισαχθεί μετά τον κόμβο που δείχνει ο
            prodeiktis. */
{
        if (keni(prodeiktis)) /*εισαγωγή στην αρχή λίστας */
                eisagogi_prin(lista,stoixeio);
        else
                eisagogi_meta(prodeiktis,stoixeio);
}
```

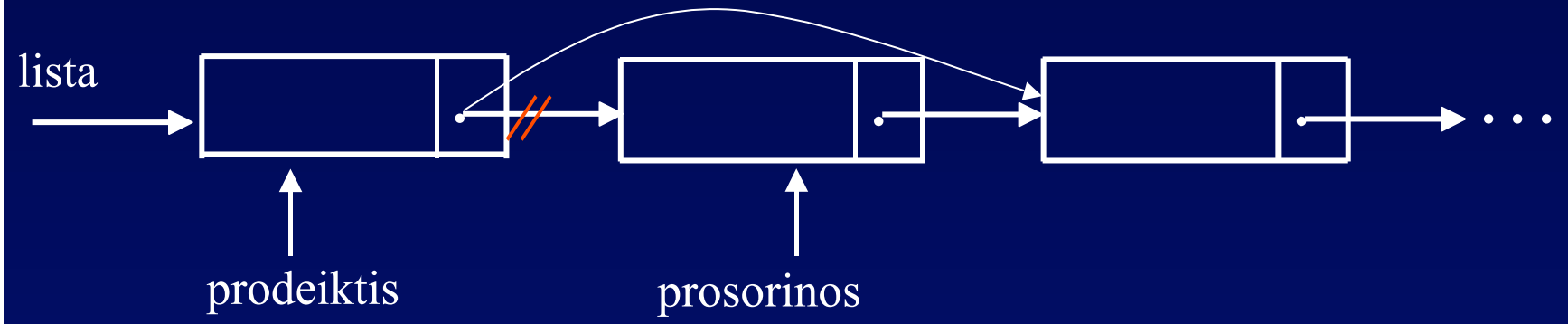
Διαγραφή

Για τη διαγραφή, υποθέτουμε ότι ο prosorinos δείχνει τον κόμβο που πρόκειται να διαγραφτεί. Υπάρχουν και εδώ δύο περιπτώσεις : (1) η διαγραφή του πρώτου στοιχείου της λίστας και (2) η διαγραφή κάποιου στοιχείου που έχει ένα προηγούμενο στοιχείο. Για τη διαγραφή του κόμβου που βρίσκεται στην αρχή της λίστας απλά μεταφέρουμε το δείκτη lista ώστε να δείχνει στο δεύτερο κόμβο της λίστας.



Για τη δεύτερη περίπτωση το ρόλο της lista παίζει ο :

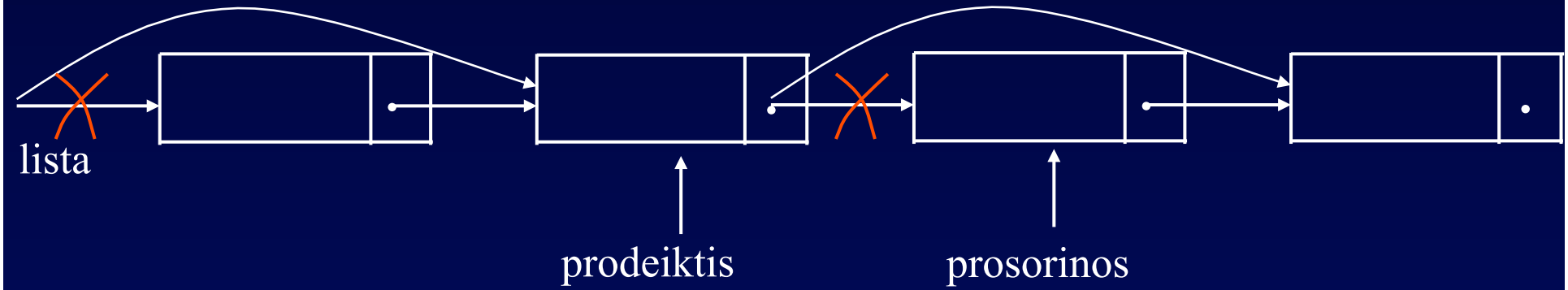
komvos[prodeiktis].epomenos



Ο παρακάτω αλγόριθμος περιγράφει τη λειτουργία της διαγραφής :

Αλγόριθμος για τη διαγραφή από μια συνδεδεμένη λίστα

Αν η λίστα είναι κενή τότε
να εκτυπωθεί ένα μήνυμα
διαφορετικά
1. Αν prodeiktis ==null τότε {διαγραφή πρώτου κόμβου}
α. prosorinos = lista
β. lista = epomenos(prosorinos)
διαφορετικά {ο κόμβος έχει προηγούμενο}
α. prosorinos = epomenos(prodeiktis)
β. epomenos(prodeiktis) = epomenos(prosorinos)
2. Apodesmeysi(prosorinos) {δημιουργία κενού χώρου στον πίνακα}



Το υποπρόγραμμα που υλοποιεί τη λειτουργία της διαγραφής είναι το ακόλουθο:

```
void diagrafi_komvou(typos_deikti *lista, typos_deikti prodeiktis)
/*Προ : Η λίστα *lista δεν είναι κενή και ο prodeiktis πρέπει να
είναι ένας κόμβος της λίστας ή να είναι null.
Μετά: Αν η *lista δεν είναι κενή και ο prodeiktis είναι null
 διαγράφεται το  πρώτο στοιχείο της λίστας. Αν η *lista δεν
 είναι κενή και ο prodeiktis είναι ένας κόμβος της λίστας
 τότε διαγράφεται  ο επόμενος κόμβος από αυτόν που δείχνει
            ο prodeiktis. */
```

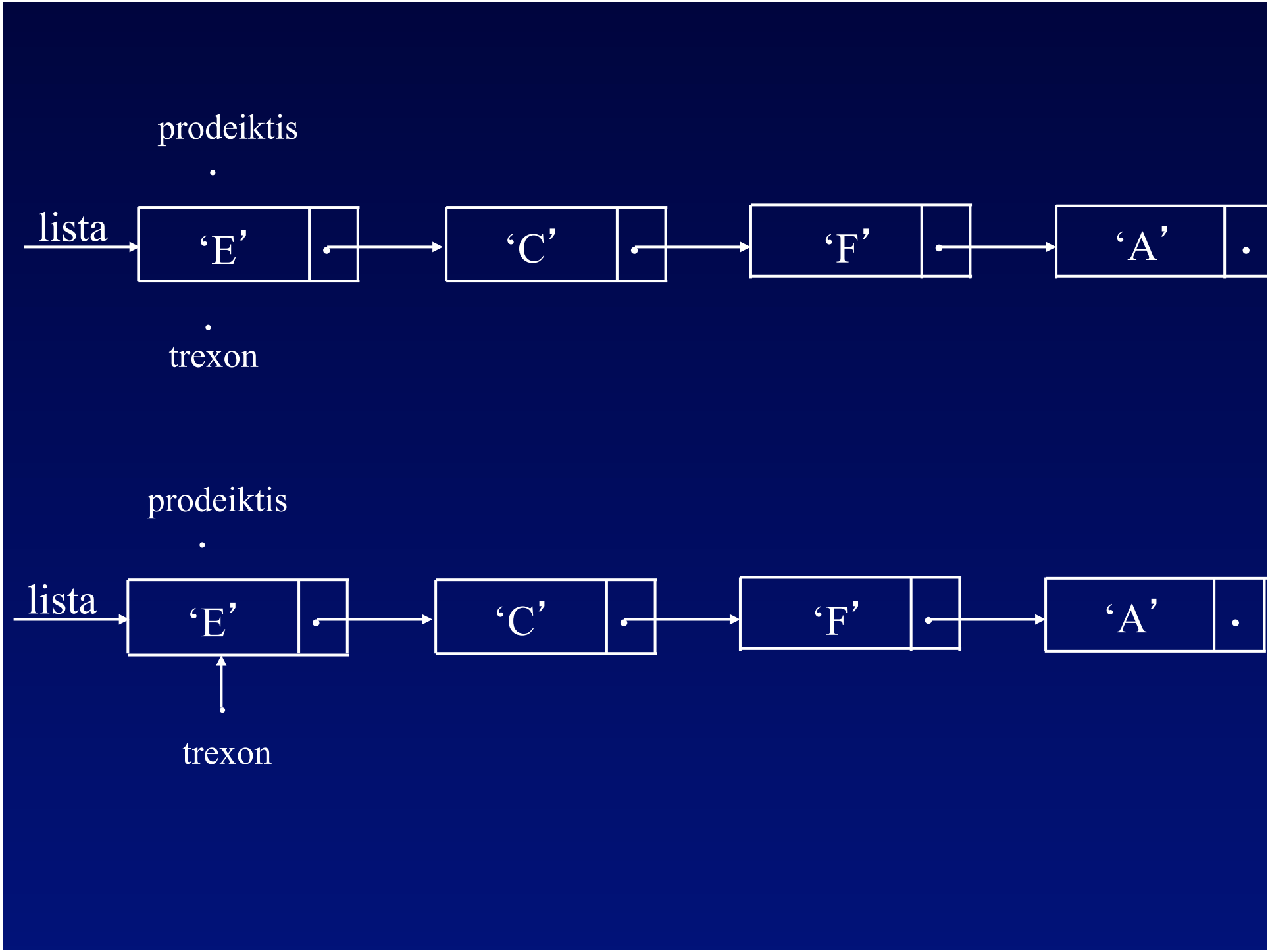
συνέχεια

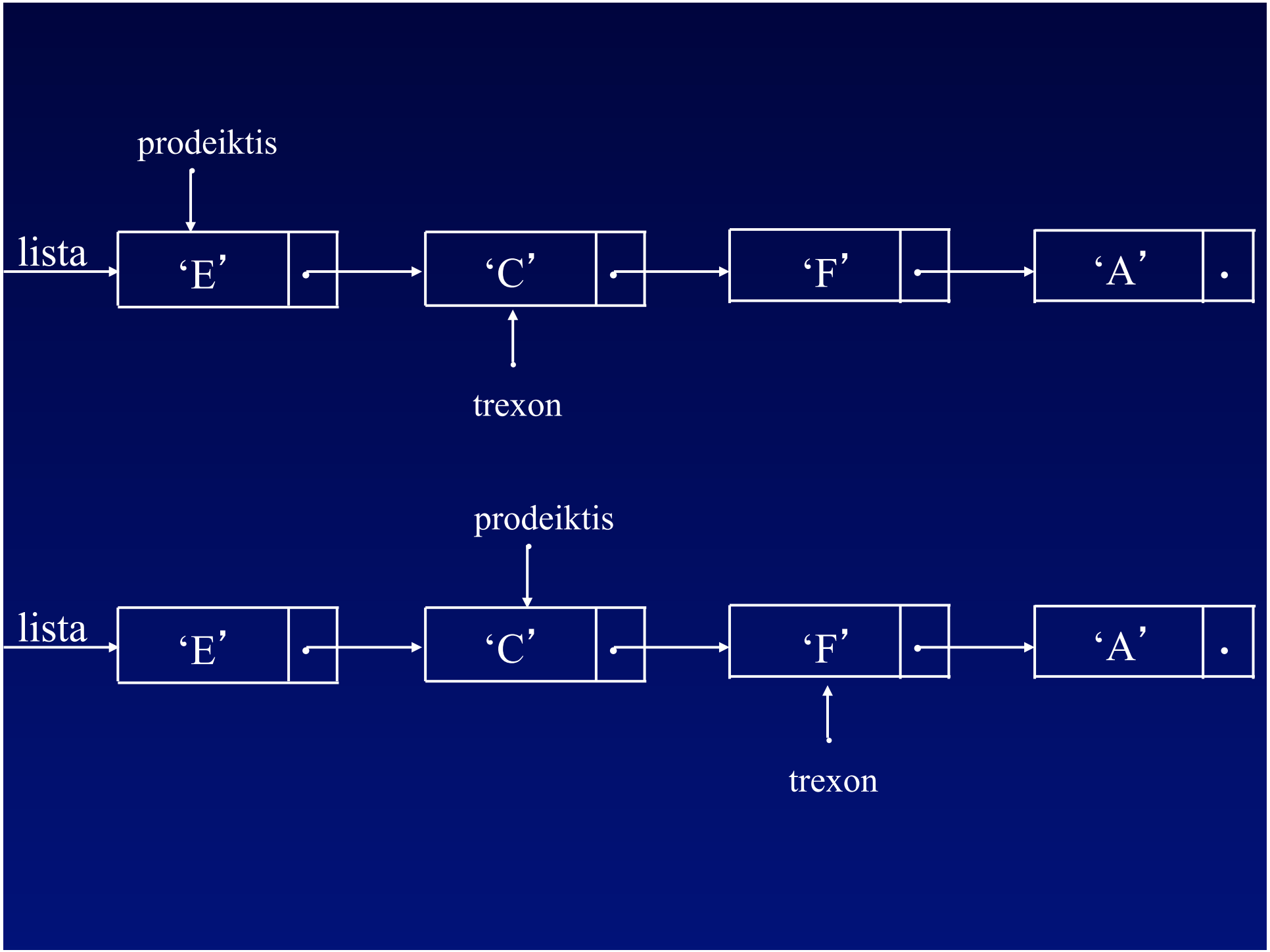
```
        if (keni(*lista))
                printf(" Κενή λίστα");
        else
        {
                if (keni(prodeiktis))
                        diagrafi(lista);
                else
                        diagrafi(&komvos[prodeiktis].epomenos);
        }
}
```

```
void diagrafi(typos_deikti *p)
/* Μετά: Διαγράφθηκε ο κόμβος που έδειχνε ο *p. Ο *p δείχνει
                    πλέον τον επόμενο κόμβο. */
{
        typos_deikti prosorinos;

        prosorinos = *p;
        *p = komvos[prosorinos].epomenos;
        apodesmeysi(prosorinos);
}
```

Αναζήτηση

Διατρέχονται όλοι οι κόμβοι της συνδεδεμένης λίστας από την αρχή ελέγχοντας το περιεχόμενο του τμήματος των δεδομένων τους. Αν αυτό συμπίπτει με το ζητούμενο, η αναζήτηση σταματά. Είναι φανερό ότι για τον εντοπισμό του στοιχείου που πρόκειται να διαγραφτεί χρειάζονται δύο δείκτες. Ο ένας δείχνει τον τρέχοντα κόμβο του οποίου εξετάζεται το περιεχόμενο και ο άλλος δείχνει τον προηγούμενο κόμβο :


prodeiktis
lista
'E'
'C'
'F'
'A'
.
trexon
prodeiktis
lista
'E'
'C'
'F'
'A'
.
trexon


prodeiktis
lista
‘E’
‘C’
‘F’
‘A’
.
trexon
prodeiktis
lista
‘E’
‘C’
‘F’
‘A’
.
trexon

Αλγόριθμος αναζήτησης σε μια συνδεδεμένη λίστα

1. prodeiktis = NULL, trexon = NULL

2. Οσο ο ζητούμενος κόμβος δεν έχει βρεθεί και ο δείκτης
   trexon !=NULL , τότε εκτελούνται οι παρακάτω εργασίες :
   Αν dedomena(trexon) = stoixeio τότε
   βρέθηκε ο ζητούμενος κόμβος
   διαφορετικά
   α. prodeiktis = trexon
   β. trexon= epomenos(trexon)

Η υλοποίηση του παραπάνω αλγόριθμου γίνεται με το ακόλουθο υποπρόγραμμα:

```
void anazitisi1(typos_deikti lista, typos_stoixeiou stoixeio,
                typos_deikti *prodeiktis, int *vrethike)
/* Μετά: Αναζήτηση στη lista για εντοπισμό του πρώτου
κόμβου
        που έχει περιεχόμενο stoixeio (*vrethike=1) ή  για μια
        θέση για την εισαγωγή νέου κόμβου (*vrethike=0) */
{
        typos_deikti trexon;

        trexon = lista;
        *prodeiktis = null;
        *vrethike = 0;
```

συνέχεια

```
        while ((!(*vrethike)) && (!keni(trexon)))
                if (periexomeno(trexon) == stoixeio)
                        *vrethike = 1;
                else
                {
                        *prodeiktis = trexon;
                         proxorise(&trexon);
                }
}
```

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 βρέθηκε ο κόμβος αυτός ή όχι.

Αλγόριθμος αναζήτησης ταξινομημένης (σε αύξουσα σειρά) λίστας

1. prodeiktis = null, trexon = null

2. Οσο δεν έγινε αναζήτηση και trexon <> null να εκτελούνται οι παρακάτω εργασίες:
   Αν dedomena(trexon) >= stoixeio, τότε:
   Η αναζήτηση έγινε και ο κόμβος βρέθηκε
   εφόσον dedomena(trexon) = = stoixeio
   διαφορετικά
   προχώρησε τους δείκτες prodeiktis και trexon.

Η υλοποίηση του παραπάνω αλγορίθμου γίνεται με την ακόλουθη διαδικασία :

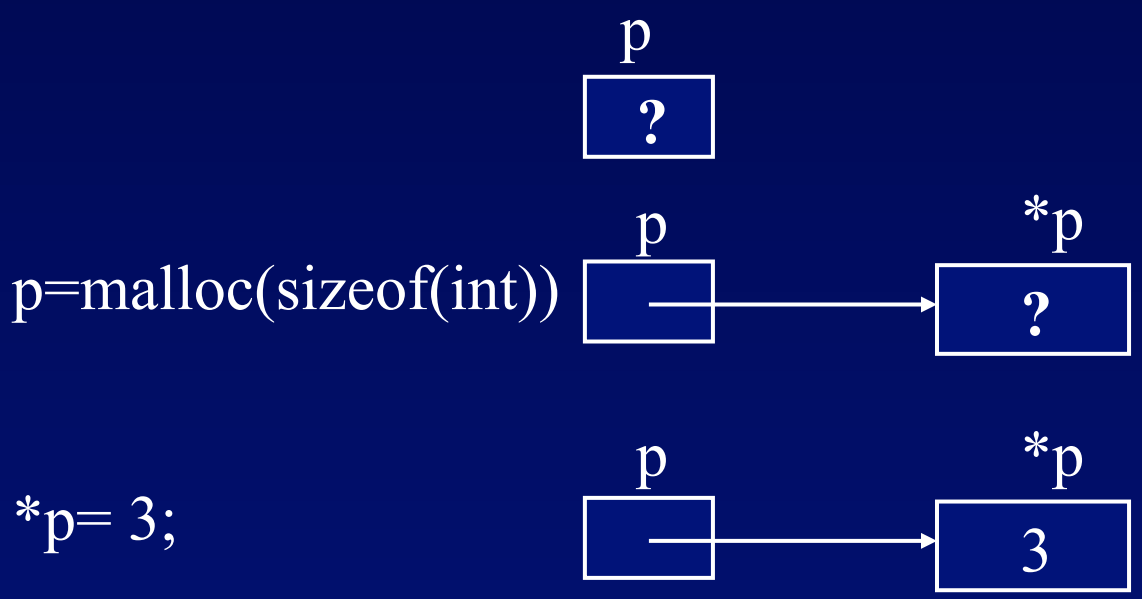
```
void anazitisi2(typos_deikti lista, typos_stoixeiou stoixeio,
                typos_deikti *prodeiktis, int *vrethike)

/* 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

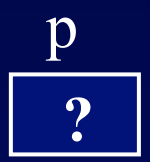
```
{       typos_deikti trexon;
        int anazitisi;
        trexon = lista;
        *prodeiktis = null;
        *vrethike = 0;
        anazitisi = 0;
        while ((!anazitisi) && (!keni(trexon)))
                if (periexomeno(trexon) >= stoixeio)
                {   anazitisi = 1;
          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

Δείκτες (Pointers) στην C

```
typedef    int    *deikths_ar;
deiktis_ar        p,q;
```



free(p)



p = NULL;

p = q;

Υλοποίηση του ΑΤΔ συνδεδεμένη λίστα με δείκτες

Α) Με ορισμό τύπου δείκτη

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

typedef struct typos_komvou
{
        typos_stoixeiou dedomena;
        typos_deikti epomenos;
};
  typos_deikti lista;
```

Ας σημειωθεί ότι ο ορισμός του τύπου δείκτη προηγείται του ορισμού του τύπου του κόμβου, πράγμα που επιτρέπεται στην C.

Υλοποίηση του ΑΤΔ συνδεδεμένη λίστα με δείκτες

Β) Με ορισμό τύπου κόμβου

```
typedef ... typos_stoixeiou ;

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

typos_komvou  *lista;
```

•Τον ρόλο των διαδικασιών pare_komvo(p) και apodesmeysi(p) τον παίζουν οι malloc(sizeof(int) και free(p), αντίστοιχα.

• Η υλοποίηση των βασικών πράξεων για τον ΑΤΔ συνδεδεμένη λίστα με τη χρήση δεικτών είναι όμοια με εκείνη του πίνακα.

• Οι διαφορές εστιάζονται στους υπολογισμούς των δεικτών.

```
void dimiourgia(typos_deikti *lista)
 /*Προ: Καμμία.
  Μετά : Εχει δημιουργηθεί η κενή λίστα lista*/
 {
        *lista = NULL;
  }
```

Ο έλεγχος μιας κενής λίστας γίνεται με το υποπρόγραμμα:

```
int keni(typos_deikti lista)
/*Προ : Εχει δημιουργηθεί η lista.
  Μετά : Επιστρέφει 1 ή 0 ανάλογα αν η lista είναι κενή  ή όχι*/

{
        return ( lista == NULL );
}
```

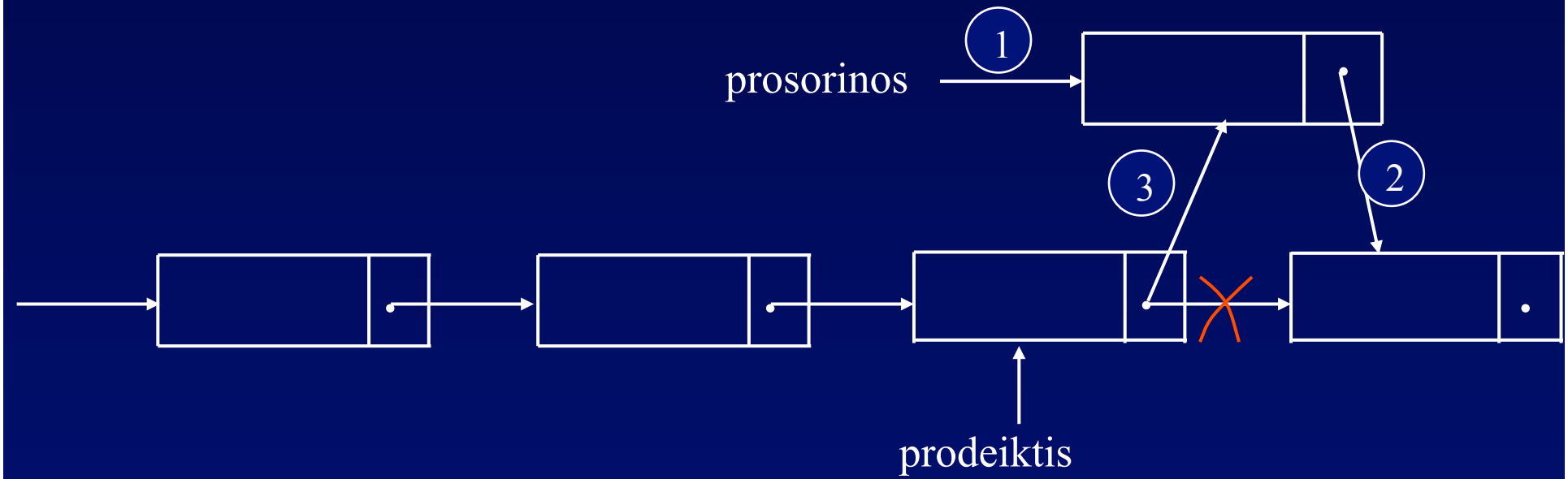
Η πράξη προχώρησε το δείκτη έτσι ώστε να δείχνει τον επόμενο κόμβο υλοποιείται ως εξής:

```
void proxorise(typos_deikti *p)
/*Προχωρά το δείκτη p στον επόμενο κόμβο της λίστας*/
{
        *p = (*p)->epomenos;
}
```

Η ανάκτηση του περιεχομένου ενός κόμβου υλοποιείται με το ακόλουθο υποπρόγραμμα :

```
typos_stoixeiou periexomeno(typos_deikti p)
/*Επιστρέφει τα δεδομένα του κόμβου που δείχνει ο δείκτης p*/
{
        return (p->dedomena);
}
```

Οι διαδικασίες για την εισαγωγή και διαγραφή στοιχείου από μια συνδεδεμένη λίστα με δείκτες είναι παρόμοιες με εκείνες που βασίζονται στην υλοποίηση με πίνακα και παρουσιάζονται στη συνέχεια :



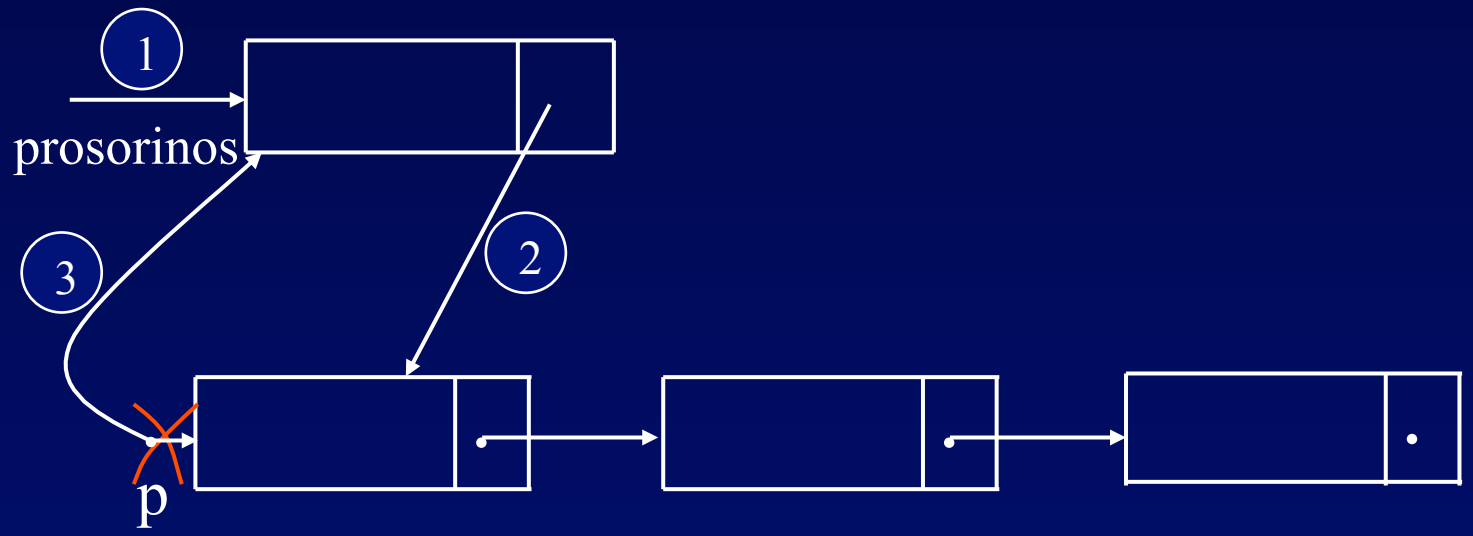
```
void eisagogi_meta(typos_deikti prodeiktis, typos_stoixeiou stoixeio)

/* Προ : Ο prodeiktis  δείχνει ένα κόμβο στη λίστα.
    Μετά: Ο κόμβος με δεδομένα stoixeio έχει εισαχθεί μετά τον κόμβο
που
    δείχνει ο δείκτης prodeiktis. */
{
    typos_deikti prosorinos;

    prosorinos = (typos_deikti) malloc(sizeof(struct typos_komvos));
    prosorinos->dedomena = stoixeio;
    prosorinos->epomenos = prodeiktis->epomenos;
    prodeiktis->epomenos = prosorinos;
}
```

1

2

3

Εισαγωγή



```
void eisagogi_prin(typos_deikti *p, typos_stoixeiou stoixeio)

/* Προ : Ο δείκτης *p δείχνει το πρώτο στοιχείο της λίστας.
    Μετά: Ο κόμβος με δεδομένα stoixeio έχει εισαχθεί  πρίν τον
κόμβο
            που έδειχνε ο δείκτης *p. Ο δείκτης *p δείχνει την αρχή της
            λίστας   */
{                                                              1
    typos_deikti prosorinos;

    prosorinos = (typos_deikti) mall 2 (sizeof(struct
typos_komvou));          3
    prosorinos->dedomena = stoixeio;
    prosorinos->epomenos = *p;
   *p = prosorinos;
}
```

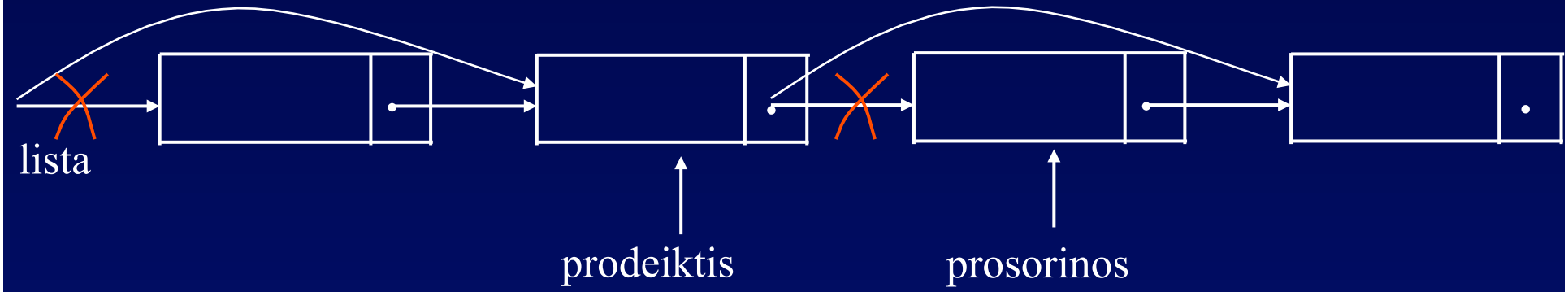
Η εισαγωγή ενός κόμβου σε οποιοδήποτε σημείο της λίστας υλοποιείται από το ακόλουθο υποπρόγραμμα.

```
void eisagogi(typos_deikti *lista, typos_stoixeiou stoixeio,
                                        typos_deikti prodeiktis)

/*Προ :Ο prodeiktis  είναι NULL ή να δείχνει ένα κόμβο μέσα στη
        λίστα.
   Μετά: Αν ο prodeiktis είναι NULL τότε ο κόμβος με δεδομένα
         stoixeio εισάγεται στην αρχή της λίστας αλλιώς εισάγεται
μετά
         τον κόμβο που δείχνει ο prodeiktis. */
```

συνέχεια

```
}
        if (keni(prodeiktis))
                eisagogi_prin(lista, stoixeio);
        else
                eisagogi_meta(prodeiktis, stoixeio);
}
```

Διαγραφή

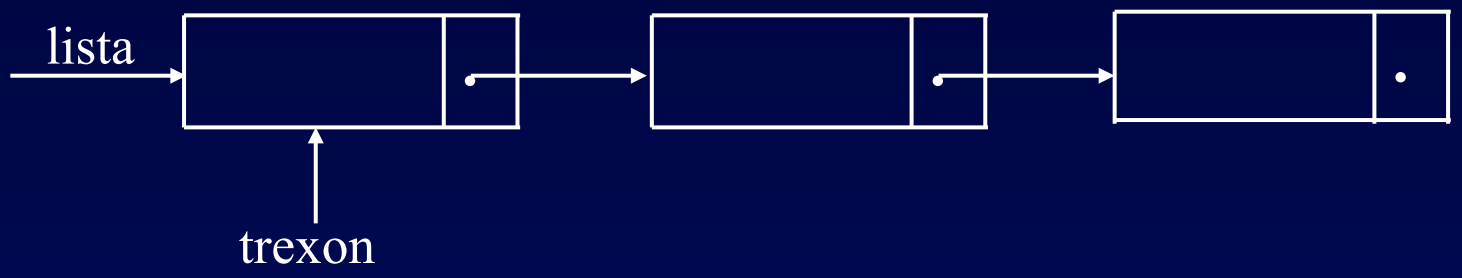


```
void diagrafi_komvou(typos_deikti *lista, typos_deikti
prodeiktis)

/*Προ : Η *lista δεν είναι κενή και ο
        prodeiktis πρέπει να είναι ένας κόμβος της
        λίστας ή να είναι NULL.

Μετά: Αν η *lista δεν είναι κενή και ο prodeiktis είναι NULL
τότε διαγράφθηκε το πρώτο στοιχείο της λίστας. Αν η *lista
δεν  είναι κενή και ο prodeiktis είναι ένας κόμβος της λίστας
τότε  διαγράφθηκε ο επόμενος κόμβος από αυτόν που δείχνει
ο prodeiktis */
```

συνέχεια

```
typos_deikti prosorinos;

 if (keni(*lista))
        printf(" Η λίστα είναι κενή");
else
{
        if (keni(prodeiktis)) /* διαγραφή στην αρχή */
                diagrafi(lista);
        else      /* ο κόμβος έχει προηγούμενο */
        {       prosorinos = prodeiktis->epomenos ;
                diagrafi(&prosorinos);
                prodeiktis->epomenos = prosorinos;
        }
}
}
```

```
void diagrafi(typos_deikti *p)
/*Προ : Ο *p δείχνει ένα κόμβο.
   Μετά: Διαγράφθηκε ο κόμβος που έδειχνε ο *p.
         Ο *p δείχνει πλέον τον επόμενο κόμβο. */
{
        typos_deikti prosorinos;

        prosorinos = *p;
        *p = prosorinos ->epomenos;
        free(prosorinos);
}
```




```
void diadromi(typos_deikti lista)
/*Προ : Εχει δημιουργηθεί μια λίστα.
   Μετά: Διέτρεξε όλους τους κόμβους μιας λίστας και
         επεξεργάστηκε τα δεδόμενα κάθε κόμβου. */
{
         typos_deikti trexon;
         trexon = lista;
         while (!keni(trexon))
         {/* εντολές για επεξεργασία του periexomeno(trexon) */
                  proxorise(&trexon);
         }
}
```

Υλοποίηση του ΑΤΔ στοίβα με συνδεδεμένη λίστα

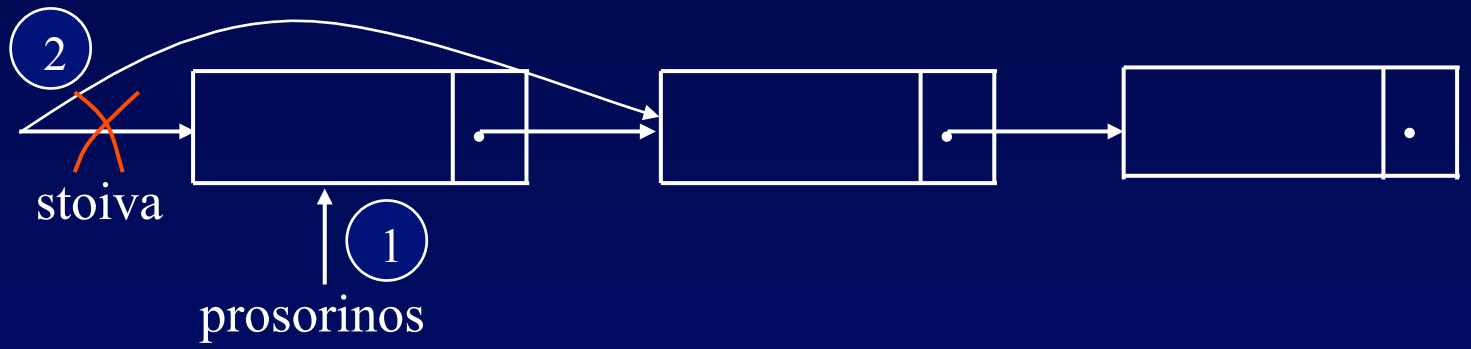
Η στοίβα υλοποιείται σαν μια συνδεδεμένη λίστα, όπου ο δείκτης που δείχνει τον πρώτο κόμβο της λίστας θα παίζει τον ρόλο της korifi.

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

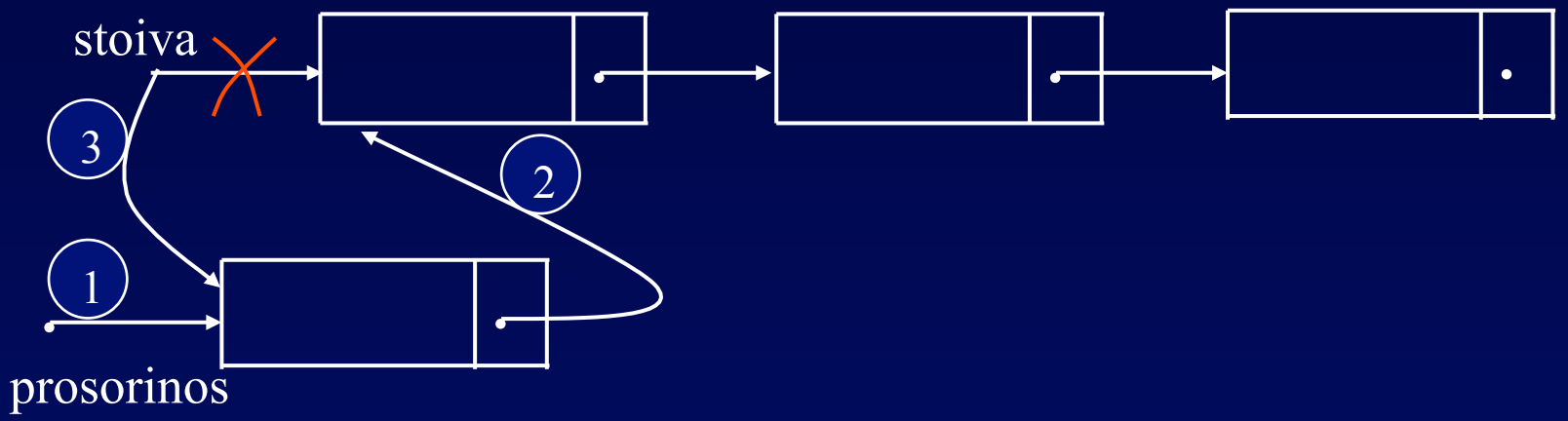
```
typedef ... typos_stoixeiou;
typedef struct typos_komvou *typos_deikti;
typedef struct typos_komvou
{
        typos_stoixeiou dedomena;
        typos_deikti epomenos;
};
typedef typos_deikti typos_stoivas;
typos_stoivas stoiva;
```

Η εξαγωγή και η ώθηση στοιχείου από τη στοίβα είναι ειδικές περιπτώσεις εισαγωγής και εξαγωγής στοιχείου από μια συνδεδεμένη λίστα.

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




```
stoixeio = dedomena(stoiva);

diagrafi(stoiva);
```



eisagogi_prin(stoiva)

```
void dimiourgia(typos_stoivas *stoiva)
{
        *stoiva = NULL;     ( dimiourgia_listas (stoiva);)
}
int keni(typos_stoivas stoiva)
{
        return ( stoiva == NULL);    ( return ( keni_listas(stoiva));)
}
void othisi(typos_stoivas *stoiva, typos_stoixeiou stoixeio)
{
        eisagogi_prin(stoiva,stoixeio);
}
```

```
void exagogi(typos_stoivas *stoiva, typos_stoixeiou *stoixeio)
{
        if (keni(*stoiva))
                printf(" Η στοίβα είναι κενή");

        else
        {
                 *stoixeio = periexomeno(*stoiva);
                diagrafi(stoiva);
                }
}
```

Η παρατήρηση εδώ είναι ότι στη διαδικασία othisi δεν υπάρχει ο έλεγχος για το αν η στοίβα είναι γεμάτη όπως στην υλοποίηση με πίνακα. Ο μόνος περιορισμός στην υλοποίηση της στοίβας με δείκτες είναι η συνολική διαθέσιμη μνήμη. Ετσι, η υλοποίηση αυτή παριστάνει με μεγαλύτερη πιστότητα τον ΑΤΔ στοίβα. Επίσης αξίζει στο σημείο αυτό να τονιστεί το γεγονός ότι τα υποπρογράμματα των βασικών πράξεων του ΑΤΔ στοίβα λειτουργούν είτε ο ΑΤΔ συνδεδεμένη λίστα έχει υλοποιηθεί με πίνακα ή με δείκτες. Με άλλα λόγια ο ΑΤΔ συνδεδεμένη λίστα χρησιμοποιείται στην παρούσα περίπτωση για την υλοποίηση ενός άλλου ΑΤΔ (στοίβα). Ακριβώς και για το λόγο αυτό δίνεται έμφαση στη χρήση του ΑΤΔ.

Στη συνέχεια παρουσιάζεται το πρόγραμμα για την μετατροπή ενός δεκαδικού μη αρνητικού ακέραιου αριθμού στο δυαδικό σύστημα. Στο πρόγραμμα αυτό έχουν τροποποιηθεί μόνο οι δηλώσεις και τα υποπρογράμματα dimiourgia, keni, exagogi και othisi σε σχέση με το προηγούμενο.

```
/* Εδώ τοποθετούνται οι Συναρτήσεις
Υλοποίησης Πράξεων Στοίβας */
main() {
/* Το πρόγραμμα αυτό μετατρέπει ένα θετικό ακέραιο
   από το δεκαδικό στο δυαδικό σύστημα και τυπώνει
   την τελική παραστασή του. */
        int arithmos,ypoloipo;
        typos_stoivas stoiva;
        printf("Δώστε ένα θετικό ακέραιο:");
        scanf("%d",&arithmos);
```

συνέχεια

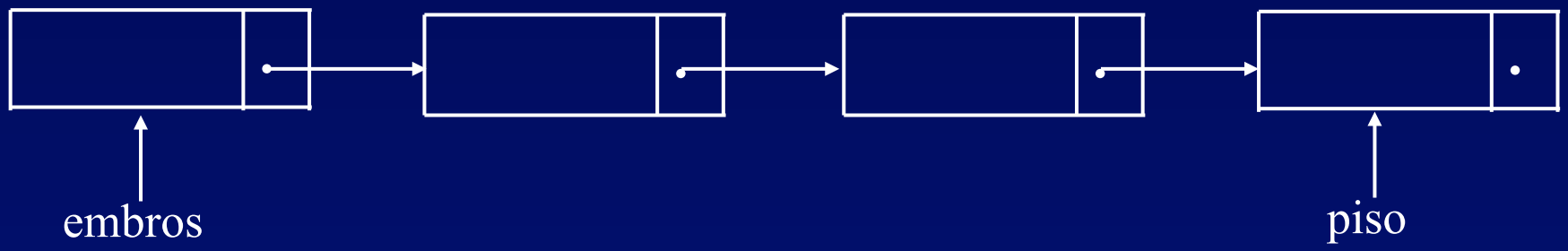
```
dimiourgia(&stoiva);
 while (arithmos)
{
        ypoloipo=arithmos % 2;
        othisi(&stoiva,ypoloipo);
        arithmos=arithmos / 2;
}
printf("Ο αριθμός στο δυαδικό σύστημα είναι: \n");
 while ( !keni(stoiva) )
{
        exagogi(&stoiva,&ypoloipo);
        printf("%d",ypoloipo);
}
printf("\n\n");
}
```

συνέχεια

Ας σημειωθεί ότι 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 τροποποιήσεις του κύριου προγράμματος. Στο σημείο αυτό παρατηρεί κανείς έντονα το πλεονέκτημα της απόκρυψης της πληροφορίας που επιτυγχάνεται ακολουθώντας τη μέθοδο του ΑΤΔ.

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.



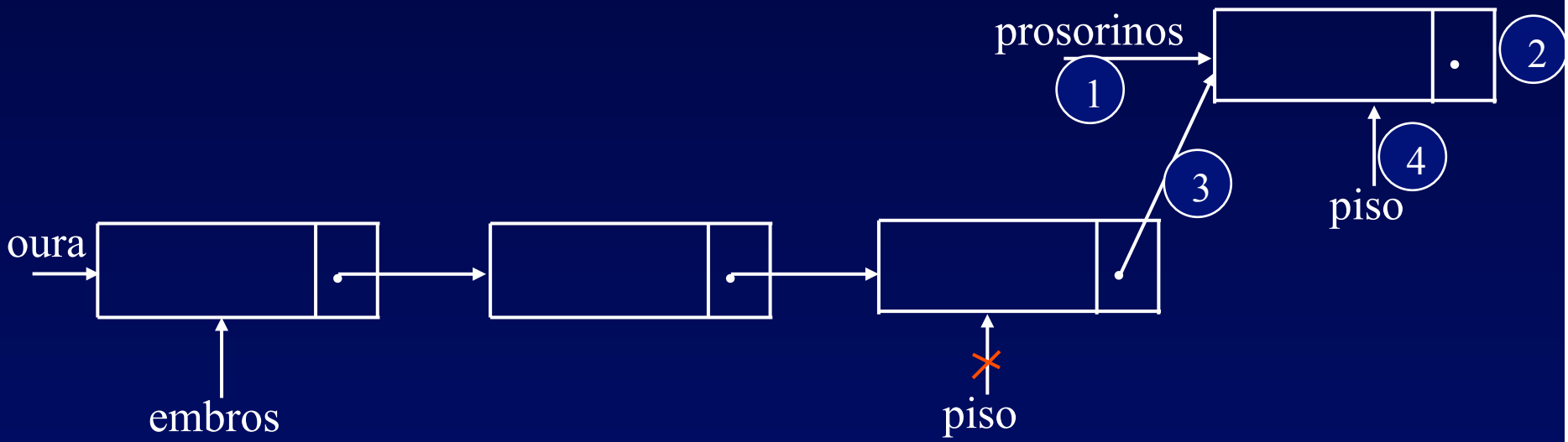
Χρειαζόμαστε τις δηλώσεις:

```
typedef  ... typos_stoixeiou ;
typedef struct komvos_ouras *typos_deikti;
typedef struct komvos_ouras
{
        typos_stoixeiou dedomena;
        typos_deikti epomenos;
};
typedef struct
{
        typos_deikti embros,piso;
} typos_ouras;
 typos_ouras oura;
```

Στη συνέχεια παρουσιάζονται τα υποπρογράμματα που αναφέρονται στις βασικές πράξεις του ΑΤΔ ουρά υλοποιημένου με συνδεδεμένη λίστα και δείκτες.

```
void dimiourgia(typos_ouras *oura)
{
        oura->embros = NULL;
        oura->piso = NULL;
}

int keni(typos_ouras oura)
{
        return (oura.embros == NULL);
}
```

Πρόσθεση στοιχείου στην ουρά




```
pare_komvo(prosorinos);   1
dedomena(prosorinos) = stoixeio;
epomenos(prosorinos) = NULL;   2
epomenos(piso) = prosorinos;   3
piso = prosorinos;   4
```

```
void prosthesi(typos_ouras *oura, typos_stoixeiou stoixeio)
{       /* Μετά: Το στοιχείο stoixeio έχει προστεθεί στο
           τέλος της ουράς *oura. */

        typos_deikti prosorinos;
        prosorinos = (typos_deikti) malloc(sizeof(struct komvos_our
        prosorinos->dedomena = stoixei
        prosorinos->epomenos = NULL;
        /* Εισαγωγή του νέου κόμβου στο πίσω μέρος της ουράς */

        if (keni(*oura))
                oura->embros = prosorinos;
        else
                oura->piso->epomenos = prosorinos;

        /* Ενημέρωση του piso ώστε να δείχνει τον τελευταίο κόμβο */
        oura->piso = prosorinos
}
```

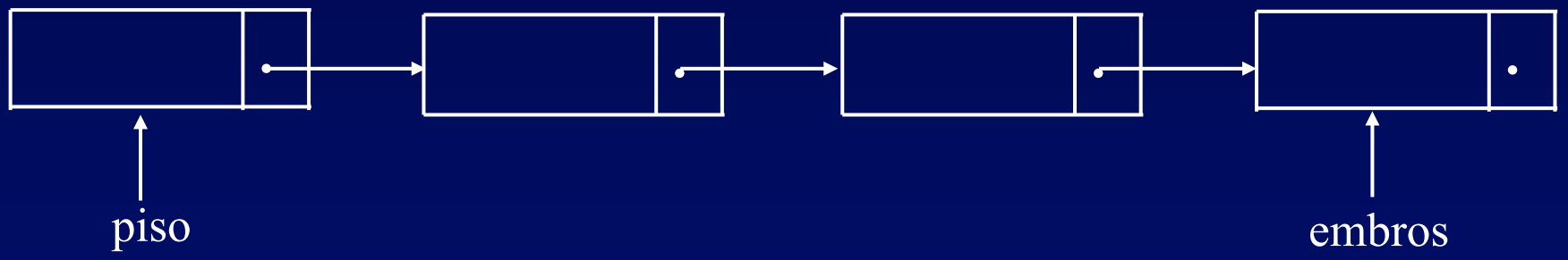
1

2

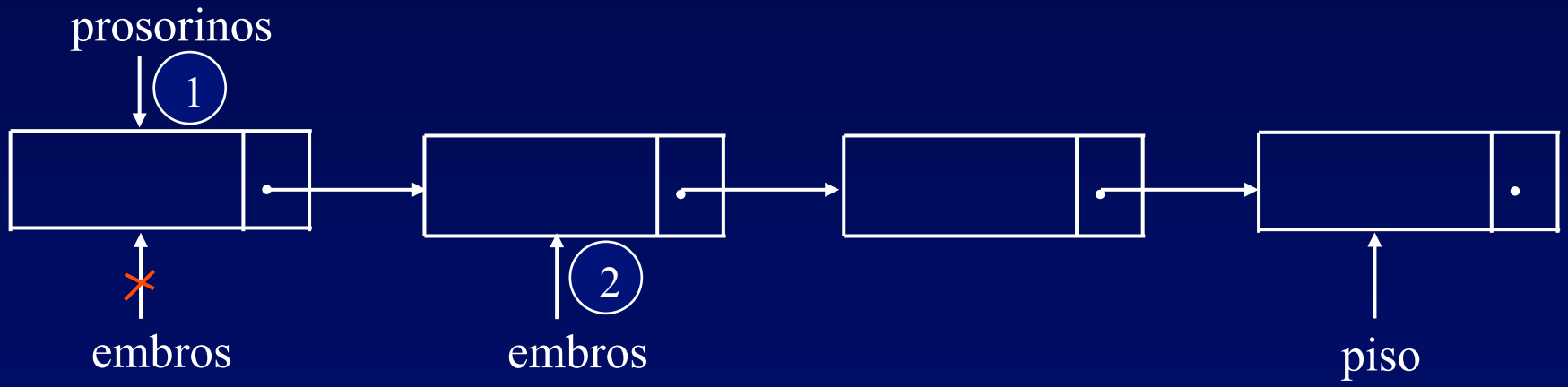
3

4

Αν η ουρά είχε παρασταθεί όπως στο παρακάτω σχήμα, όπου οι δείκτες embros και piso έχουν ανταλλάξει θέσεις, τότε θα υπήρχε πρόβλημα μετά τη διαγραφή του κόμβου.



Απομάκρυνση στοιχείου από την ουρά




```
prosorinos = embros;   (1)
stoixeio = dedomena(embros);
embros = epomenos(embros);   (2)
apodesmeysi(prosorinos);
```

```
void apomakrinsi(typos_ouras *oura, typos_stoixeiou *stoixeio)
/*Μετά: Αν η *oura δεν είναι κενή τότε διαγράφεται το πρώτο
στοιχείο της  *oura και αποθηκεύεται στο *stoixeio και
επιστρέφεται 1, αλλιώς επιστρέφεται 0. */
{       typos_deikti prosorinos;
        if (keni(*oura))
                printf(" Η ουρά είναι κενή");
        else
        {
              *stoixeio = periexomeno(oura->embros);
              diagrafi(oura->embros);
             if keni(oura) /* Αν η ουρά είναι κενή τότε piso=NULL*/
                    piso = NULL;
         }
}
```

Αναζήτηση κόμβου

Ας υποθέσουμε ότι έχουμε μια ταξινομημένη συνδεδεμένη λίστα και θέλουμε να εντοπίσουμε ένα στοιχείο της με συγκεκριμένο περιεχόμενο στο τμήμα των δεδομένων του ή ότι θέλουμε να εντοπίσουμε τη θέση στην οποία θα εισάγουμε ένα νέο κόμβο.

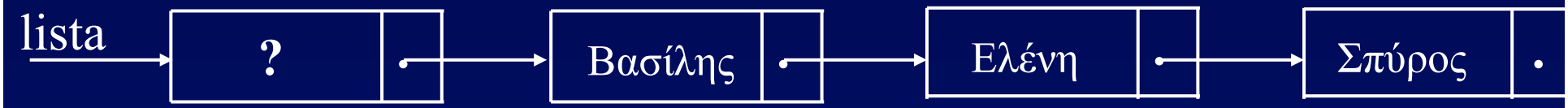
```
void anazitisi_listas(typos_deikti lista, typos_stoixeiou stoixeio,
                      typos_deikti *prodeiktis, int *vrethike)
{/*Μετά: Έγινε αναζήτηση της lista για τον εντοπισμό κόμβου
που έχει περιεχόμενο stoixeio (στην περίπτωση αυτή
*vrethike=1) ή για μια θέση για την εισαγωγή ενός νέου κόμβου
(*vrethike= 0).Το *prodeiktis δείχνει τον προηγούμενο κόμβο
από αυτόν που περιέχει το stoixeio ή τον κόμβο μετά τον οποίο
μπορεί να εισαχθεί το stoixeio */
```

συνέχεια

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

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.



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

Η δημιουργία μιας κενής συνδεδεμένης λίστας με κεφαλή περιγράφεται από το υποπρόγραμμα:

```
typedef ... typos_stoixeiou ;
typedef struct typos_komvou *typos_deikti;
typedef struct typos_komvou
{       typos_stoixeiou dedomena;
        typos_deikti epomenos;
};
 typos_deikti lista;

void dimiourgia(typos_deikti *lista)
{
   *lista = (typos_deikti) malloc(sizeof(struct typos_komvou));
    (*lista)->epomenos = NULL;
}
```

Ομοια για τον έλεγχο αν μια λίστα είναι κενή έχουμε:

```
int keni(typos_deikti lista)
/*Ελέγχει αν μια λίστα με κεφαλή είναι κενή*/
{
        return (lista->epomenos == NULL);
}
```

Οι πράξεις proxorise και περιεχόμενο παραμένουν οι ίδιες. Τόσο οι αλγόριθμοι όσο και τα αντίστοιχα υποπρογράμματα για τις πράξεις της εισαγωγής και διαγραφής απλοποιούνται για τις συνδεδεμένες λίστες με κεφαλή. Ετσι το υποπρόγραμμα eisagogi τροποποιείται στο παρακάτω:

```
void eisagogi (typos_stoixeiou stoixeio, typos_deikti prodeiktis)
/* Μετά: Ο κόμβος με δεδομένα stoixeio έχει εισαχθεί στη
συνδεδεμένη λίστα με κεφαλή μετά τον κόμβο που δείχνει ο
prodeiktis. */
 {
        typos_deikti prosorinos;
        prosorinos = (typos_deikti) malloc(sizeof(struct typos_komvou));
        prosorinos->dedomena= stoixeio;
        prosorinos->epomenos= prodeiktis->epomenos;
        prodeiktis->epomenos= prosorinos;}
```

συνέχεια

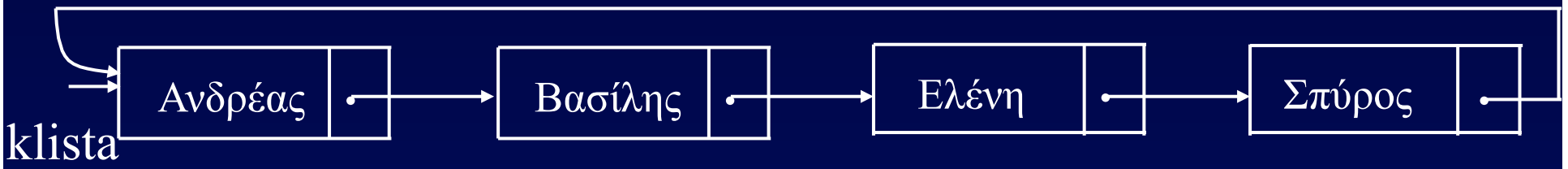
Ομοια για το υποπρόγραμμα diagrafi έχουμε:

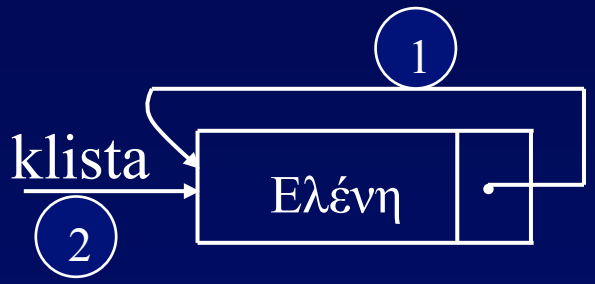
```
void diagrafi(typos_deikti *lista, typos_deikti prodeiktis)
/*Προ : Η *lista δεν είναι κενή.
   Μετά: Αν η λίστα *lista δεν είναι κενή τότε διαγράφθηκε
         από τη *lista ο επόμενος κόμβος από αυτόν που δείχνει ο
prodeiktis  */
{
         typos_deikti prosorinos;

         if (keni(*lista))
                  printf(" Η λίστα είναι κενή");
```

συνέχεια

```
        else
        {
                prosorinos = prodeiktis->epomenos;
                prodeiktis->epomenos =prosorinos->epomenos;
                free(prosorinos);
        }
}
```

Κυκλικά Συνδεδεμένες Λίστες





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_komvo(prosorinos)

2. dedomena(prosorinos)= stoixeio

3. Αν η κυκλική λίστα είναι κενή τότε

α. epomenos(prosorinos) = prosorinos

β. klista = prosorinos

διαφορετικά

α. epomenos(prosorinos)= epomenos(prodeiktis)

β. epomenos(prodeiktis) = prosorinos

Το αντίστοιχο υποπρόγραμμα είναι το παρακάτω:

```
void eisagogi_klista(typos_deikti *klista, typos_stoixeiou stoixeio,
                                                    typos_deikti prodeiktis)
{/* Μετά: Ο κόμβος με δεδομένα stoixeio έχει εισαχθεί(στην
κυκλικά συνδεδεμένη λίστα *klista) μετά τον κόμβο που δείχνει ο
prodeiktis. */

        typos_deikti prosorinos;

        prosorinos = (typos_deikti) malloc(sizeof(struct typos_komvou));
        prosorinos->dedomena = stoixeio;
        if (keni_klista(*klista))
        {       prosorinos->epomenos = prosorinos;
                *klista = prosorinos;
        }
```

συνέχεια

```
        else
        {
                prosorinos->epomenos = prodeiktis->epomenos;
                prodeiktis->epomenos = prosorinos;
        }
}
```

Αλγόριθμος διαγραφής σε κυκλικά συνδεδεμένη λίστα

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

Αν η λίστα είναι κενή τότε
να τυπωθεί ένα μήνυμα
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 diagrafi_klista(typos_deikti *klista, typos_deikti prodeiktis)
/* Προ : Η λίστα *klista δεν είναι κενή.
Μετά: Αν η *klista δεν είναι κενή τότε διαγράφθηκε ο επόμενος
κόμβος από αυτόν που δείχνει ο prodeiktis */
 {
        typos_deikti prosorinos;

         if (keni_klista(*klista))
                printf(" Η λίστα είναι κενή");
        else
        {
                prosorinos = prodeiktis->epomenos;
```

```
            if (prosorinos == prodeiktis)          /* μόνο ένας κόμβος */
                    *klista = NULL;
            else
             {
                    prodeiktis->epomenos =prosorinos->epomenos;
                    if (prosorinos == *klista)
                    /* Διαγραφή 1ου κόμβου */
                    *klista = prosorinos->epomenos;
            }
        free(prosorinos);
    }
}
```

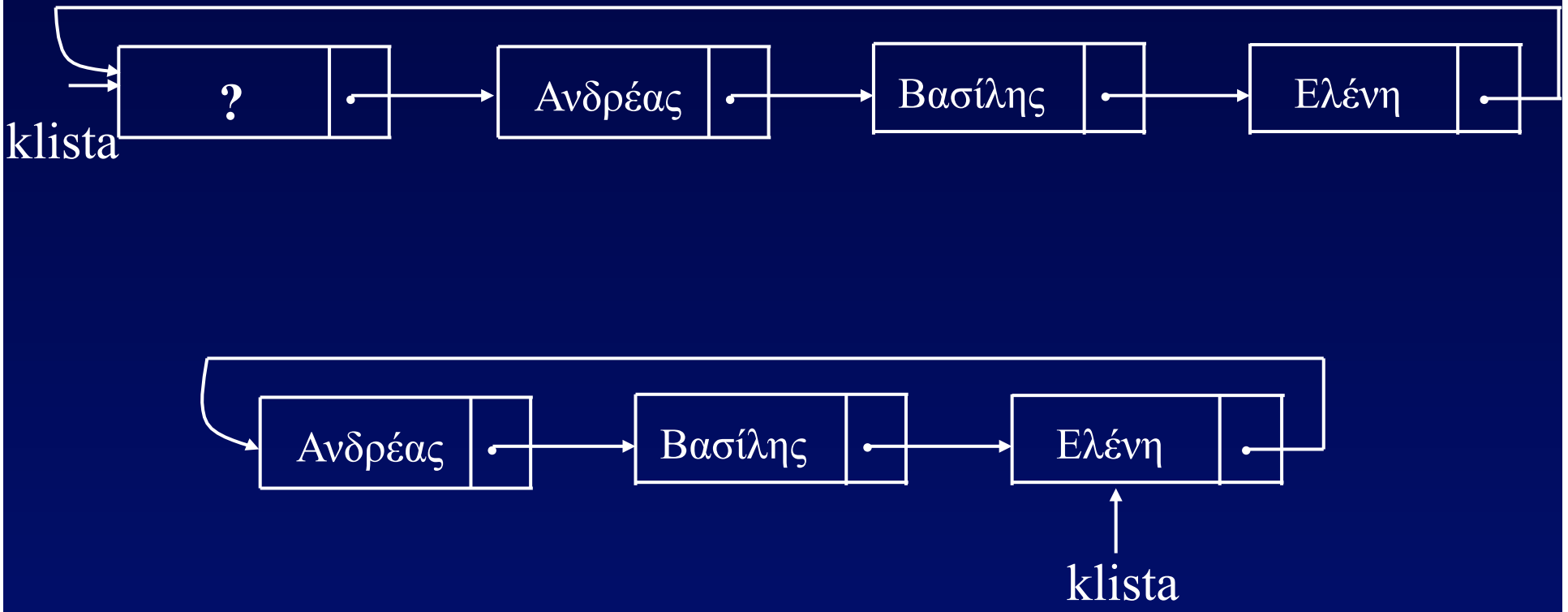
Αλγόριθμος για την επίσκεψη των κόμβων μιας κυκλικά συνδεδεμένης λίστας

Αν η λίστα δεν είναι κενή τότε:

1. trexon = klista

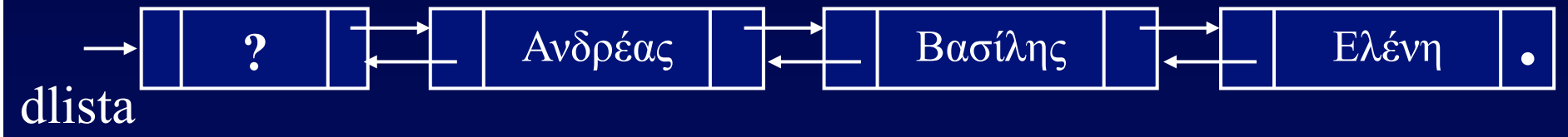
2. Να επαναλαμβάνονται τα ακόλουθα βήματα:
α. Επεξεργασία του dedomena(trexon)
β. trexon = epomenos(trexon)
μέχρις ότου trexon == klista

Κυκλικά Συνδεδεμένη Λίστα με Κεφαλή

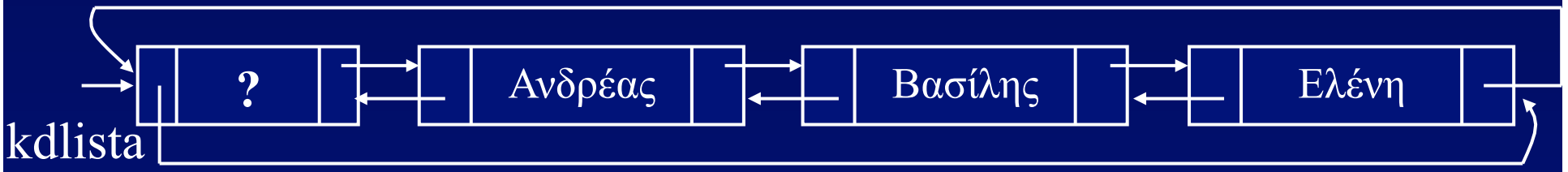


Διπλά Συνδεδεμένες Λίστες

Διπλά Συνδεδεμένες Λίστες :



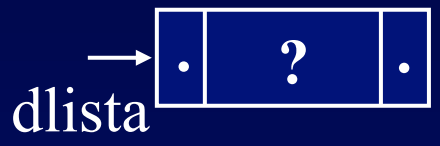
Κυκλικά Διπλά Συνδεδεμένες Λίστες :



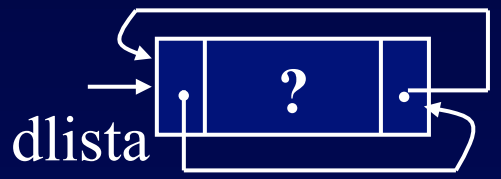
Για την υλοποίηση των διπλά συνδεδεμένων λιστών χρειάζονται οι ακόλουθοι ορισμοί και δηλώσεις:

```
typedef ... typos_stoixeiou ;
typedef struct typos_komvou *typos_deikti;
typedef struct typos_komvou
{
        typos_stoixeiou dedomena;
        typos_deikti    epomenos,proigoumenos;
};

typos_deikti dlista;
```

Οι αλγόριθμοι για τις βασικές πράξεις με διπλά συνδεδεμένες λίστες είναι όμοιοι με εκείνους των συνδεδεμένων λιστών μιας κατεύθυνσης.

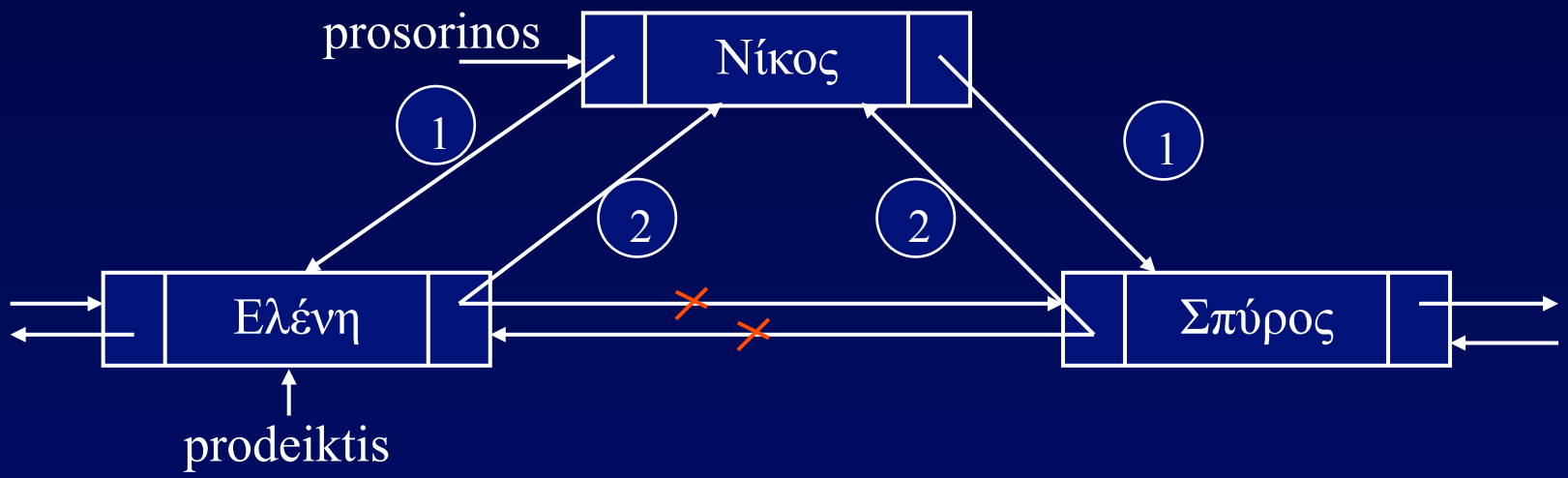



```
void dimiourgia_dlistas(typos_deikti       *dlista)
/*Δημιουργεί μια κενή διπλά συνδεδεμένη λίστα με κεφαλή*/
{
        *dlista = (typos_deikti) malloc(sizeof(struct typos_komvou));
        (*dlista)->epomenos = NULL;
        (*dlista)->proigoumenos = NULL;
}
```




```
int keni_dlista(typos_deikti dlista)
/*Ελέγχει αν μια διπλά συνδεδεμένη λίστα με κεφαλή είναι
κενή*/

{
        return (dlista->epomenos == dlista->proigoumenos);
}
```

Εισαγωγή κόμβου σε μια διπλά συνδεδεμένη λίστα με κεφαλή

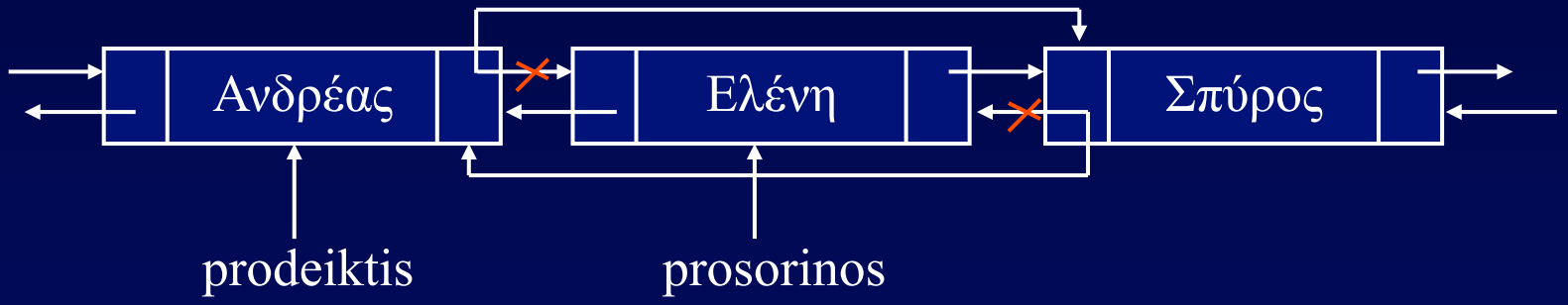


Πιο συγκεκριμένα, πρέπει να ακολουθηθούν τα επόμενα βήματα:

1. Δημιουργία κόμβου στον οποίο να δείχνει ο δείκτης prosorinos.
2. Καταχώρηση της τιμής στο τμήμα των δεδομένων του, δηλαδή το όνομα Νίκος.
3. Σύνδεση του κόμβου με τη λίστα.

```
void eisagogi_dlista(typos_stoixeiou stoixeio,typos_deikti
prodeiktis)
/* Μετά: Ο κόμβος με δεδομένα stoixeio έχει εισαχθεί στη διπλά
συνδεδεμένη λίστά με κεφαλή μετά τον κόμβο που δείχνει ο
prodeiktis. */
 {
      typos_deikti   prosorinos;
      prosorinos = (typos_deikti) malloc(sizeof(struct
typos_komvou));
      prosorinos->dedomena = stoixeio;
      prosorinos->proigoumenos = prodeiktis;
      prosorinos->epomenos = prodeiktis->epomenos;
      prodeiktis->epomenos = prosorinos;

      if (prosorinos->epomenos != NULL)
          prosorinos->epomenos->proigoumenos =prosorinos;
}
```

Διαγραφή κόμβου σε μια συνδεδεμένη λίστα με κεφαλή



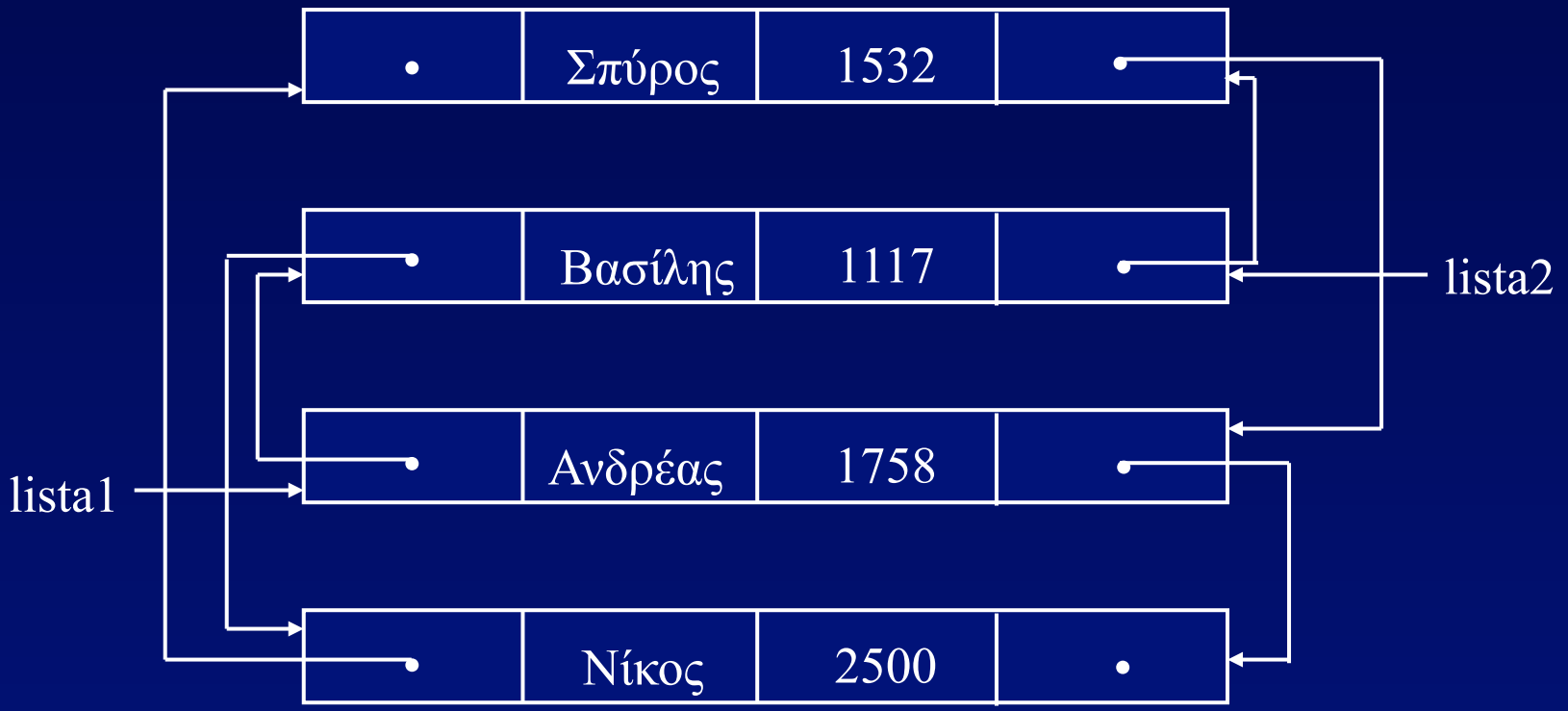

```
void diagrafi_dlista(typos_deikti *dlista,typos_deikti prodeiktis)

/* Προ : Η διπλά συνδεδεμένη λίστά με κεφαλή δεν είναι κενή.
Μετά: Αν η *dlista δεν είναι κενή τότε διαγράφθηκε ο επόμενος
κόμβος από αυτόν που δείχνει ο prodeiktis  */

        typos_deikti prosorinos;
```

συνέχεια

```
        if (keni_dlista(*dlista))
                printf("Κενή λίστα");
        else
        {
            prosorinos = prodeiktis->epomenos;
            prodeiktis->epomenos = prosorinos->epomenos;
            if (prosorinos->epomenos != NULL)
                  prosorinos->epomenos->proigoumenos = prodeiktis;
            free(prosorinos);
        }
}
```

Πολλαπλά συνδεδεμένες λίστες (multiply linked lists)

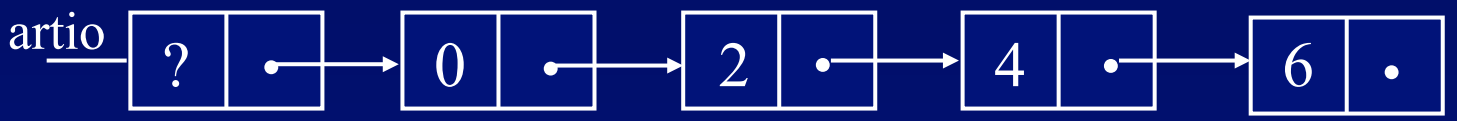
Για το προηγούμενο παράδειγμα των εγγραφών, που περιέχουν το όνομα και τον αριθμό μητρώου φοιτητών, θα μπορούσαμε να έχουμε την παρακάτω συνδεδεμένη λίστα:



Εφαρμογές συνδεδεμένων λιστών

Υλοποίηση του ΑΤΔ σύνολο με συνδεδεμένη λίστα

Η υλοποίηση του ΑΤΔ σύνολο με πίνακα επιβάλλει τον περιορισμό ότι όλα τα σύνολα, ακόμα και εκείνα με λίγα μόνο στοιχεία, παριστάνονται με πίνακες των οποίων το μέγεθος είναι ίσο με το μέγεθος του καθολικού συνόλου, το οποίο μπορεί να είναι αρκετά μεγάλο. Ενας άλλος περιορισμός είναι ότι τα στοιχεία του καθολικού συνόλου πρέπει να είναι ταξινομημένα γιατί πρέπει να αντιστοιχούν στους δείκτες του πίνακα. Ενας άλλος τρόπος υλοποίησης που δεν έχει τους παραπάνω περιορισμούς είναι εκείνος που χρησιμοποιεί τη συνδεδεμένη λίστα για την παράσταση ενός συνόλου. Για παράδειγμα, το σύνολο των αρτίων ψηφίων θα μπορούσε να παρασταθεί με την ακόλουθη συνδεδεμένη λίστα με κεφαλή:



Ας υποτεθεί ότι έχουμε τις δηλώσεις:

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

typos_deikti synolo;
```

Με βάση τις παραπάνω δηλώσεις η δημιουργία και ο έλεγχος μιας κενής συνδεδεμένης λίστας με κεφαλή υλοποιούνται από τα ακόλουθα υποπρογράμματα :

```
void dimiourgia(typos_deikti *synolo)
{
        *synolo = (typos_deikti) malloc(sizeof(struct typos_komvou));
        (*synolo)->epomenos = NULL;
}
```

```
int keno(typos_deikti synolo)
/*Προ: Δημιουργία κενής συνδεδεμένης λίστας με κεφαλή.
   Μετά : Επιστρέφει 1 ή 0 ανάλογα αν είναι κενή λίστα ή όχι*/
{
        return (synolo->epomenos == NULL);
}
```

Για την υλοποίηση της εισαγωγής ενός νέου στοιχείου σε μια συνδεδεμένη λίστα με κεφαλή, χρησιμοποιείται το ακόλουθο υποπρόγραμμα:

```
void eisagogi(typos_deikti *synolo, typos_stoixeiou x)
/* Μετά: Εισήχθη ένας νέος κόμβος με δεδομένα x
    στην αρχή της λίστας που παριστάνει το συνόλο. */
{
        typos_deikti prosorinos;
        prosorinos = (typos_deikti) malloc(sizeof(struct
typos_komvou));
        prosorinos->dedomena = x;
        prosorinos->epomenos = (*synolo)->epomenos;
        (*synolo)->epomenos = prosorinos;
}
```

Επίσης και οι άλλες πράξεις υλοποιούνται εξίσου εύκολα. Για παράδειγμα, ο έλεγχος του αν ένα στοιχείο ανήκει σε ένα σύνολο υλοποιείται με την παρακάτω συνάρτηση, η οποία στην ουσία διατρέχει τους κόμβους της λίστας.

```
int melos(typos_deikti synolo,typos_stoixeiou x)
{/* Μετά: Αν το στοιχείο x είναι μέλος του synolo τότε η
        συνάρτηση επιστρέφει 1, αλλιώς επιστρέφει 0. */

        typos_deikti trexon;
        int evrika;

        trexon = synolo->epomenos; /* έχει κεφαλή */
        evrika = 0;
```

συνέχεια

```
        while ( !evrika && (trexon != NULL) )
        {
            if (trexon->dedomena == x)
                evrika = 1;
            else
                trexon = trexon->epomenos;
        }
        return evrika;
}
```

Για την ένωση C = A ∪ B δύο συνόλων A και B, που παριστάνονται με αυτή την υλοποίηση, αντιγράφονται οι κόμβοι της A στην C και στη συνέχεια, διατρέχονται οι κόμβοι της B και αντιγράφονται στη C εφόσον δεν ανήκουν στην A.

```
typos_deikti enosi(typos_deikti A,typos_deikti B)
{/* Μετά: Η συνάρτηση επιστρέφει την ένωση των συνόλων
        A και B. */

        typos_deikti C,prosorinos;
        dimiourgia(&C);
        /* αντιγραφή του A στο C */
        prosorinos = A;
        proxorise(&prosorinos);
```

συνέχεια

```
while (!keno(prosorinos))
{       eisagogi(&C,periexomeno(prosorinos));
        proxorise(&prosorinos);
}
/* αντιγραφή των στοιχείων του B,που δεν ανήκουν στο
A,στο C */
prosorinos = B;
proxorise(&prosorinos);
while(!keno(prosorinos))
{       if (!melos(A,periexomeno(prosorinos)))
        eisagogi(&C,periexomeno(prosorinos));
        proxorise(&prosorinos);
}
return C;}
```

συνέχεια

Η υλοποίηση των υπολοίπων πράξεων της διαφοράς και της τομής δύο συνόλων αφήνονται σαν άσκηση.

Υλοποίηση του ΑΤΔ συμβολοσειρά με συνδεδεμένη λίστα

Η μετατροπή της υλοποίησης με πίνακα του ΑΤΔ συμβολοσειρά σε αντίστοιχη με συνδεδεμένη λίστα είναι απλή. Αρκεί ο πίνακας των χαρακτήρων να αντικατασταθεί με μια συνδεδεμένη λίστα στην οποία ο κάθε κόμβος περιέχει ένα χαρακτήρα. Οι δηλώσεις και η υλοποίηση της συνένωσης δύο συμβολοσειρών παρουσιάζονται παρακάτω :

```
typedef struct typos_komvou *typos_deikti;
typedef struct typos_komvou
        {
                char xar;
                 typos_deikti epomenos;
        };
typedef struct
        {
               int mikos;
              typos_deikti  lista;
        } typos_seiras;

typos_seiras seira;
```

```
void synenosi(typos_seiras seira1,typos_seiras seira2,
                    typos_seiras *nea_seira)
{/* Μετά: Δημιουργήθηκε μια νέα συμβολοσειρά, η
    *nea_seira, που αποτελεί την συνένωση των
     συμβολοσειρών seira1 και seira2. */

        typos_deikti trexon;
        dimiourgia(nea_seira);
        nea_seira->mikos =          mikos(seira1)+mikos(seira2);
        /* αντιγραφή της seira1 στη nea_seira */
        trexon = (seira1.lista)->epomenos;
        while (trexon != NULL)
        {       prosartisi(nea_seira,trexon->xar);
                trexon = trexon->epomenos;
        }
```

συνέχεια

```
        /* αντιγραφή της seira2 στη nea_seira */
        trexon = (seira2.lista)->epomenos;
        while (trexon != NULL)
        {       prosartisi(nea_seira,trexon->xar);
                trexon = trexon->epomenos;
        }
}
```

Η υλοποίηση των υπολοίπων πράξεων του ΑΤΔ συμβολοσειρά αφήνεται σαν άσκηση.

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

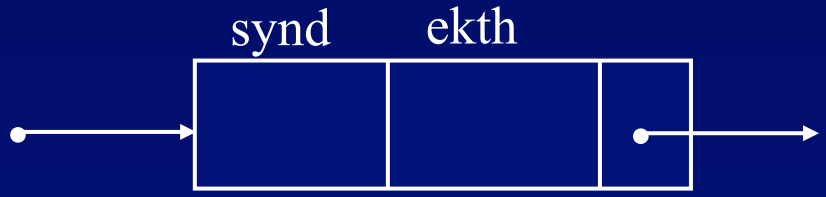
$$p(x) = 6 + 0x + 0x^2 + \ldots + 1x^{50} + \ldots + 1x^{99}$$

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.

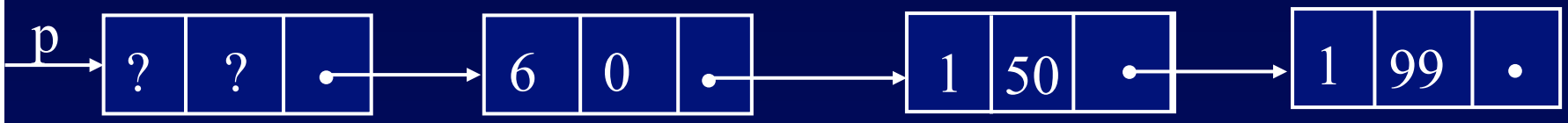
Για παράδειγμα το p(x) παριστάνεται σαν

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

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:



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

```
typedef float typos_stoixeiou ;
typedef struct typos_komvou *typos_deikti;
typedef struct typos_komvou
{
        typos_stoixeiou   synd;
      int                 ekth;
      typos_deikti        epomenos;
};
 typos_deikti p;
```

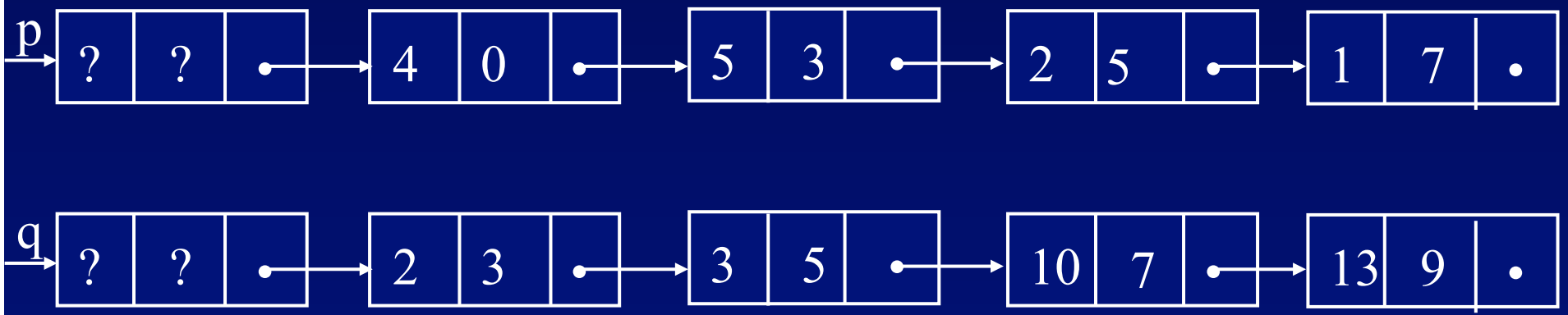
Η επεξεργασία πολυωνύμων που παριστάνονται με τον παραπάνω τρόπο δεν παρουσιάζει δυσκολίες. Για παράδειγμα, ας υποτεθεί ότι επιθυμούμε να προσθέσουμε τα πολυώνυμα

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

και

$$q(x) = 2x^3 + 3x^5 + 10x^7 - 3x^8 + 13x^9$$

που παριστάνονται σαν



Θα χρησιμοποιηθούν δύο βοηθητικοί δείκτες ptrexon και qtrexon που θα διατρέχουν τα στοιχεία των λιστών p και q, αντίστοιχα. Σε κάθε βήμα, οι ptrexon και qtrexon θα δείχνουν κόμβους που πρόκειται να επεξεργαστούν. Η επεξεργασία αρχίζει με την σύγκριση των εκθετών των δύο κόμβων που δείχνουν οι ptrexon και qtrexon. Αν οι εκθέτες είναι διαφορετικοί, τότε ο κόμβος που περιέχει το μικρότερο τοποθετείται στη νέα λίστα r που θα παριστάνει το άθροισμα των πολυωνύμων p(x) και q(x). Αν όμως οι εκθέτες είναι ίσοι, τότε οι συντελεστές αυτών των κόμβων προστίθενται και το αποτέλεσμα τοποθετείται στο πεδίο synd ενός νέου κόμβου. Στο πεδίο ekth του κόμβου αυτού τοποθετείται ο κοινός εκθέτης και ο κόμβος αυτός τοποθετείται στη λίστα r. Αν το άθροισμα των συντελεστών είναι μηδέν, τότε δεν δημιουργείται νέος κόμβος. Αν συναντηθεί 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(typos_deikti p,typos_deikti q, typos_deikti *r)
{       /* Προ : Έχουν δημιουργηθεί δύο λίστες με κεφαλές που
                  παριστάνουν δύο πολυώνυμα p και q.
           Μετά: Το *r είναι δείκτης σε μια συνδεδεμένη λίστα
           με κεφαλή που παριστάνει το άθροισμα των
           πολυωνύμων p και q. */

           typos_deikti ptrexon,qtrexon,rtrexon,prosorinos;
           typos_stoixeiou neos_synd;
```

συνέχεια

```
ptrexon = p->epomenos;
qtrexon = q->epomenos;
dimiourgia(r);
rtrexon = *r;

while ((ptrexon!=NULL) && (qtrexon!=NULL))
{
   if (ptrexon->ekth < qtrexon->ekth)
   {
       eisagogi_telos(ptrexon->synd,ptrexon->ekth,&rtrexon);
        proxorise(&ptrexon);
    }
    else
```

συνέχεια

```
if (qtrexon->ekth < ptrexon->ekth)
{
        eisagogi_telos(qtrexon->synd,
        qtrexon->ekth,&rtrexon);
        proxorise(&qtrexon);
}
else
{
neos_synd = ptrexon->synd + qtrexon->synd;
 if (neos_synd != 0)
     eisagogi_telos(neos_synd,ptrexon->ekth,&rtrexon);
 proxorise(&ptrexon);
 proxorise(&qtrexon);
}
```

συνέχεια

```
        }
        /* Αντιγραφή υπόλοπων κόμβων της p ή της q στην r */
        if (ptrexon!=NULL)
                prosorinos = ptrexon;
        else
                prosorinos = qtrexon;
        while (prosorinos!=NULL)
        {       eisagogi_telos(prosorinos->synd,
                prosorinos->ekth,&rtrexon);
                proxorise(&prosorinos);
        }
        rtrexon->epomenos = NULL;
}
```

Το υποπρόγραμμα αυτό χρησιμοποιεί το υποπρόγραμμα eisagogi_telos για τη δημιουργία κόμβου και την προσάρτησή του στο τέλος της νέας συνδεδεμένης λίστας, όπως φαίνεται παρακάτω:

```
void eisagogi_telos(typos_stoixeiou sy, int ek, typos_deikti *telos)
{       /* Προ : Έχει δημιουργηθεί μία λίστα και ο δείκτης
           *telos δείχνει τον τελευταίο κόμβο της.
           Μετά: Έχει εισαχθεί στο τέλος της λίστας ένας νέος
           κόμβος με συντελεστή sy και εκθέτη ek. */

           typos_deikti prosorinos;

           prosorinos = (typos_deikti)

           malloc(sizeof(struct typos_komvou));

           prosorinos->synd = sy;

           prosorinos->ekth = ek;
```

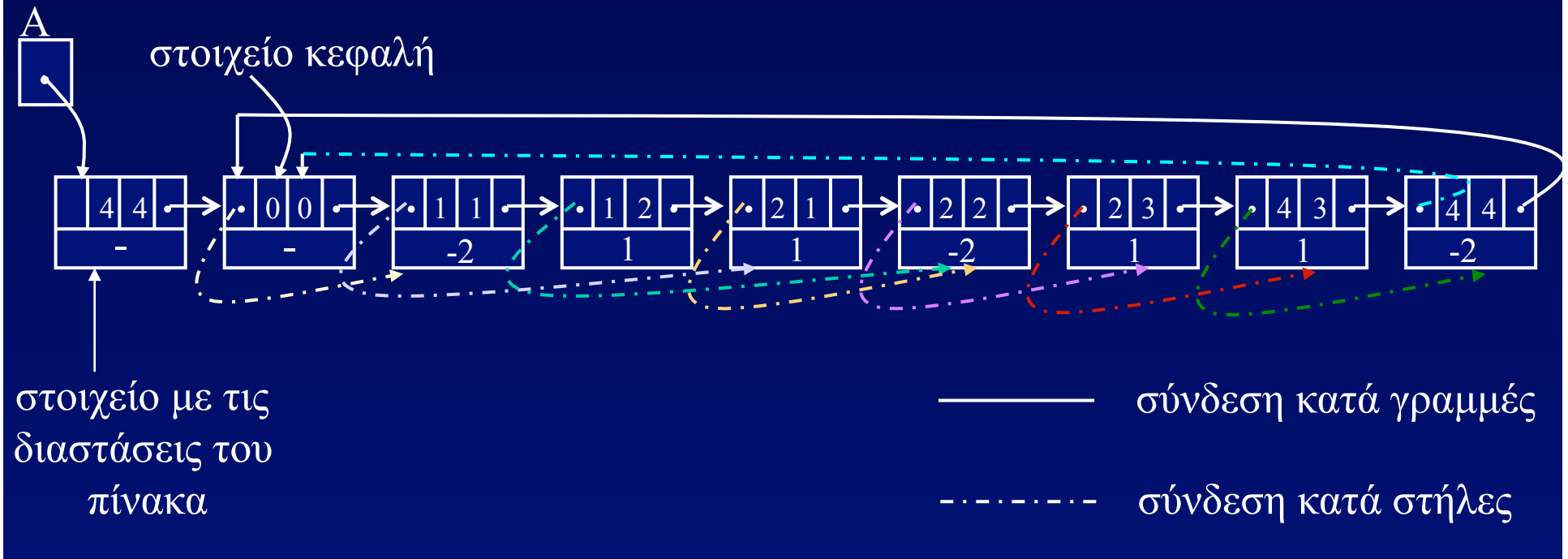
συνέχεια

```
        prosorinos->epomenos = NULL;
        (*telos)->epomenos = prosorinos;
        *telos = prosorinos;
}
```

Παράσταση αραιού πίνακα με συνδεδεμένη λίστα

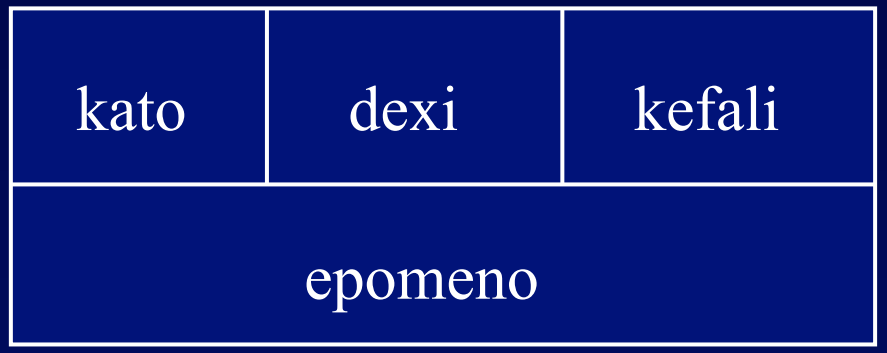
Οταν περιοριστούμε στην πρόσθεση, αφαίρεση και τον πολ/μό αραιών πινάκων μπορούμε να χρησιμοποιήσουμε μια απλούστερη και πιο αποτελεσματική συνδεδεμένη δομή.

Απλουστευμένη σύνδεση του πίνακα Α :

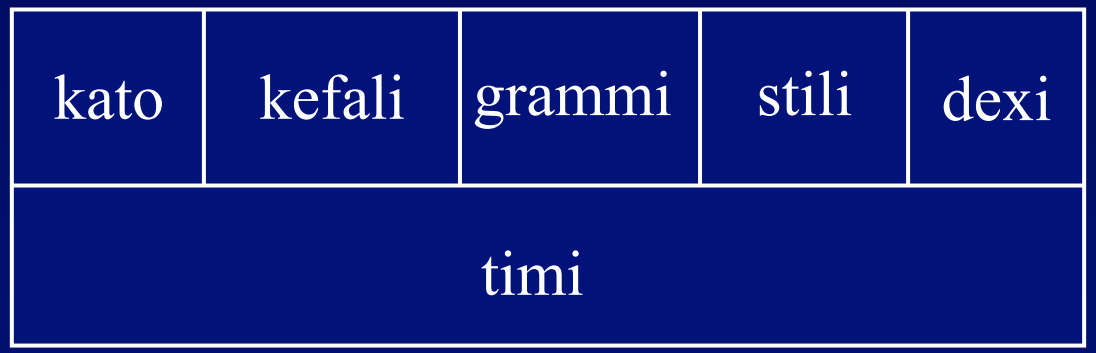


Δομή δεδομένων για αραιούς πίνακες (a_{ij} <> 0)

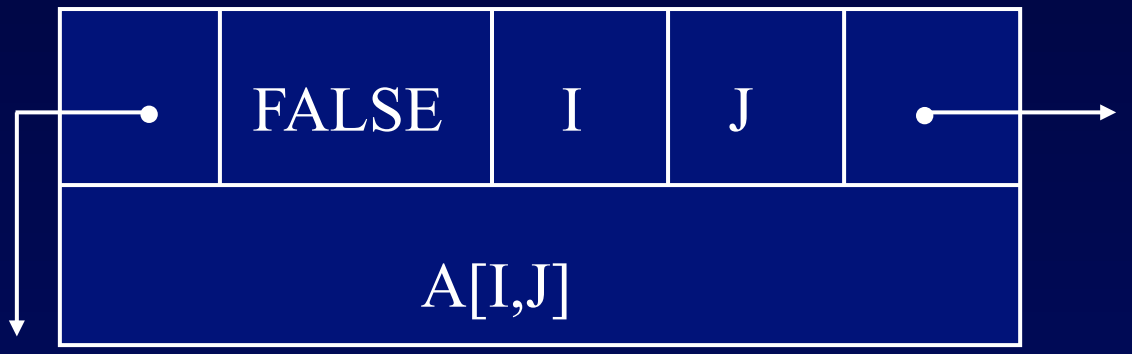
(i) Στοιχείο κεφαλή



(ii) Τυπικό στοιχείο λίστας



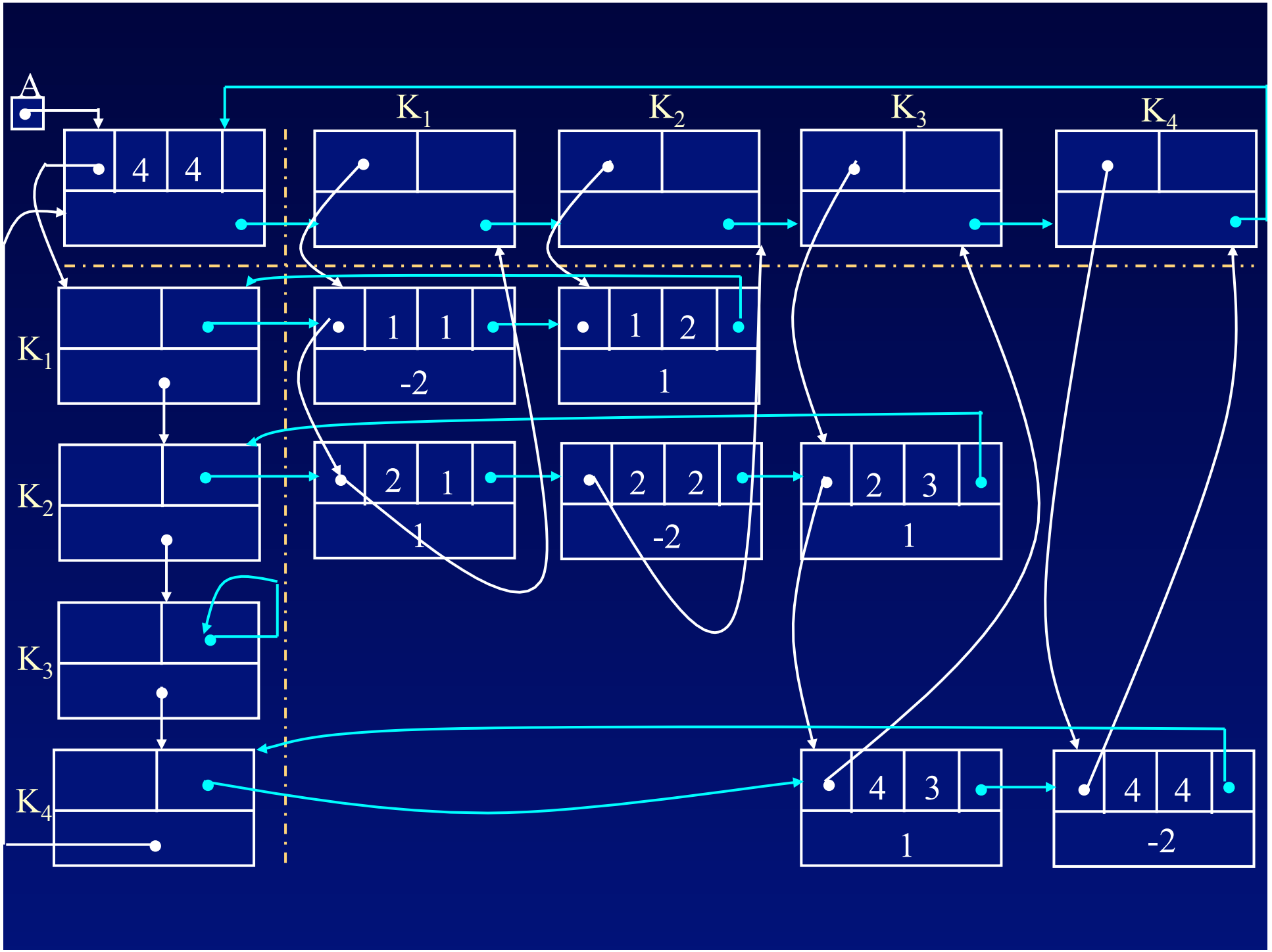
(iii) Στοιχεία για $a_{ij} <> 0$



Αραιός Πίνακας

$$A = \begin{bmatrix} -2 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 \end{bmatrix}$$

Ακολουθεί η συνδεδεμένη δομή του πίνακα Α :


A
K1
K2
K3
K4
4 4
1 1
-2
1 2
1
K1
2 1
1
2 2
-2
2 3
1
K2
K3
4 3
1
4 4
-2
K4

Πρόβλημα

Γράψτε ένα υποπρόγραμμα διαδικασία που να διαβάζει ένα αραιό πίνακα και να δημιουργεί τη συνδεδεμένη μορφή του.

Ανάλυση

Θα υποθέσουμε ότι η πρώτη γραμμή του αρχείου εισόδου αποτελείται από τον αριθμό των γραμμών n, τον αριθμό των στηλών m και τον αριθμό r των μη μηδενικών στοιχείων του πίνακα. Στη συνέχεια έπονται r γραμμές εισόδου όπου στην κάθε μία υπάρχει η τριάδα (i, j, a_{ij}) με $a_{ij} <> 0$. Επίσης υποθέτουμε ότι οι τριάδες αυτές είναι διατεταγμένες κατά γραμμές και σε κάθε γραμμή κατά στήλες.

Στο κύριο πρόγραμμα υποθέτουμε ότι υπάρχουν οι ορισμοί:

```
type
   deiktispin =  stoixeio;
   stoixeio = record
                     kato : deiktispin;
                     dexi : deiktispin;
                     case kefali : boolean of
                        true : (epomeno : deiktispin);
                        false : (timi : integer; grammi : integer;
                                 stili : integer)
                end;
var
    kefstoixeio : array [1..K] of deiktispin;
```

όπου K = max(n, m).

```
procedure diabasmapin (var A: deiktispin);
{Διάβασε ένα πίνακα και δημιούργησε τη συνδεδεμένη δομή του}
var
    i, m, n, p, r, ggrammi, sstili, ttimi, trexgrammi: integer;
    x, trexon : deiktispin;
begin
{Διαστάσεις πίνακα}
    readln (n, m, r);
    if m > n then
        p := m
    else
        p := n;
    {αρχικές τιμές στους κόμβους κεφαλές}
```

συνέχεια

```
for i := 1 to p do
    begin
        new(x);
        kefstoixeio[i] := x;
        with x  do
            begin
                dexi := x;
                epomeno := x;
                kefali := true
            end {with}
    end; {for}
trexgrami := 1;
{τρέχον κόμβος στη τρέχουσα γραμμή}
trexon := kefstoixeio[1];
```

συνέχεια

```
for i := 1 to r do
    begin
        readln (ggrammi, sstili, ttimi)
        if ggrami > trexgrami then
            begin
            {κλείσε την τρέχουσα γραμμή}
                trexon^.dexi := kefstoixeio[trexgrami];
                trexgrami := ggrami;
                trexon := kefstoixeio[ggrami]
            end;
            {στοιχείο για νέα τριάδα}
            new(x);
            with x  do
                begin
                    kefali := false;
                    grami := ggrami;
```

συνέχεια

```
                stili := sstili;
                timi := ttimi
            end;
        {σύνδεση με τη λίστα γραμμής}
        trexon^.dexi := x;
        trexon := x;
        {σύνδεση με τη λίστα στήλης}
        kefstoixeio[sstili]^.epomeno .kato := x;
        kefstoixeio[sstili]^.epomeno := x;
end; {for}
{κλείσιμο τελευταίας γραμμής}
for i := 1 to m do
if r > 0 then
   trexon^.dexi := kefstoixeio[trexgrami];
{κλείσιμο όλων των λιστών στήλης}
kefstoixeio[i]^.epomeno^.kato := kefstoixeio[i];
```

συνέχεια

```
{Δημιουργία κεφαλής της λίστας κεφαλής κόμβων}
new(a);
a^.grami := n;
a^.stili := m;
{Σύνδεση των κόμβων κεφαλές μεταξύ τους}
for i := 1 to p - 1 do
    kefstoixeio[i]^.epomeno := kefstoixeio[i+1];
    if p = 0 then
        a^.dexi := a
    else
        begin
            kefstoixeio[p]^.epomeno := a;
            a^.dexi := kefstoixeio[1]
        end
end;
```