- Τύπος δεδομένων (data type) μιας μεταβλητής (σε μια γλώσσα προγραμματισμού) είναι το σύνολο των τιμών που μπορεί να πάρει η μεταβλητή.
- Αφηρημένος τύπος δεδομένων (abstract data type): είναι ένα θεωρητικό (μαθηματικό) μοντέλο αποθήκευσης πληροφορίας μαζί με τους ορισμούς και τους αλγόριθμους κάποιων βασικών πράξεων (π.χ. αποθήκευση, διαγραφή, επεξεργασία, κλπ.).
- Η υλοποίηση ενός αφηρημένου τύπου δεδομένων σε κάποια γλώσσα προγραμματισμού λέγεται δομή δεδομένων (data structure). Η δομή δεδομένων περιέχει την αναπαράσταση του μοντέλου σε κάποιο τύπο δεδομένων της γλώσσας προγραμματισμού και την υλοποίηση των αλγόριθμων των βασικών πράξεων.

ΑΦΗΡΗΜΕΝΟΣ ΤΥΠΟΣ ΔΕΔΟΜΕΝΩΝ
(Abstract Data Type)

- ορισμός του μοντέλου αποθήκευσης των δεδομένων

π.χ. ακέραιος, πίνακας, γραμμική λίστα, σύνολο, δέντρο, γράφημα, κλπ.

- σύνολο πράξεων

π.χ. εισαγωγή, διαγραφή, εκτύπωση όλων των δεδομένων, εύρεση πληροφορίας, κλπ.

- Η έννοια του ΑΤΔ είναι θεωρητική και έχει σαν σκοπό την περιγραφή του μοντέλου αποθήκευσης των δεδομένων και των βασικών πράξεων με τις οποίες μπορούμε να επιδράσουμε σε αυτό το μοντέλο αγνοώντας τις λεπτομέρειες υλοποίησης του.

Υλοποίηση του ΑΤΔ

- υλοποίηση σε κάποια γλώσσα προγραμματισμού της δήλωσης η οποία ορίζει μια μεταβλητή του ΑΤΔ
- υλοποίηση των αλγόριθμων των βασικών πράξεων

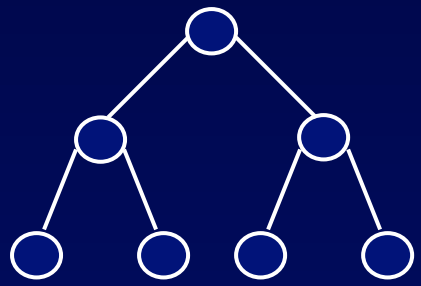
Πλεονεκτήματα του ΑΤΔ

- Επεξεργασία δεδομένων σε ένα λογικό (θεωρητικό) επίπεδο
- Απόκρυψη πληροφορίας (information hidding, encapsulation)

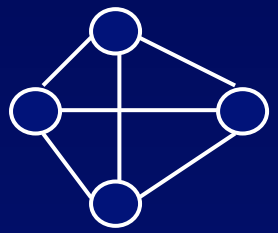
δομική σχέση στοιχείων ΑΤΔ

- γραμμική (linear)

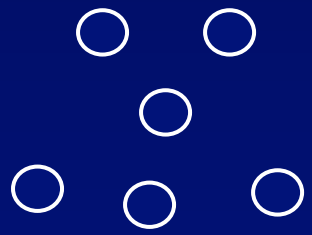
- ιεραρχική (hierarchical)



- δίκτυο (network)



- σύνολο (set)



ΑΤΔ ακέραιος (int)

Βασικές πράξεις :

καταχώρηση, πρόσθεση, αφαίρεση, πολλαπλασιασμός, διαίρεση, υπόλοιπο, μηδέν, θετικός.

Σύνολα τιμών : Ακέραιοι Αριθμοί

Παραδείγματα

επόμενος(i)

επόμενος ⟵ πρόσθεση(1, i)

αντίθετος(i)

αντίθετος ⟵ αφαίρεση(0, i)

$$|x| = \begin{cases} x, & \text{αν} \quad x >= 0 \\ -x, & \text{αν} \quad x < 0 \end{cases}$$

Αν θετικός (x) τότε

απόλυτη ⟵ x

διαφορετικά

απόλυτη ⟵ αντίθετος (x)

Υλοποίηση του ΑΤΔ ακέραιος

επίπεδο μηχανής

$$(101)_2 = 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = (5)_{10}$$

0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

16 bits

$$(111\ldots1)_2 = 2^{n-1} + 2^{n-2} + \ldots + 2^0 = 2^n - 1$$

$$n = 16 \qquad 2^{16} - 1 = 65.535$$

Ο ΑΤΔ Ακέραιος δεν παριστάνεται τέλεια στην C (και στις άλλες γλώσσες προγραμματισμού).

Στην C η υλοποίηση του ΑΤΔ είναι η ακόλουθη :

```
void kataxorisi (int *stoixeio, int timh)
/*Καταχωρεί στην ακέραια μεταβλητή stoixeio
        την πληροφορία timh*/
{
  *stoixeio=timh;
}


int prosthesi (int i, int j)
/*Προσθέτει τους ακεραίους i και j*/
{
  return(i+j);
}
```

```
int afairesi (int i, int j)
/*Αφαιρεί τον j από τον i*/
{
    return(i-j);
}


int polmos (int i, int j)
/*Πολλαπλασιάζει τους i και j*/
{
    return(i*j);
}
```

```
int diairesi (int i, int j)
/*Διαιρεί τον i δια του j   */
{
   return (i/j);
}


int ypoloipo (int i, int j)
/*Υπολογίζει το ακέραιο υπόλοιπο της διαιρέσεως i δια j */
{
   return (i % j);
}
```

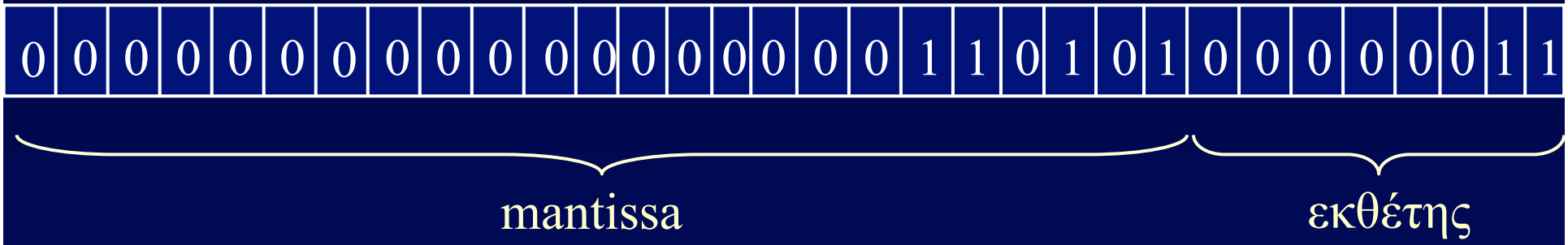
```
int miden (int i)
/*Ελέγχει αν i είναι ίσο με μηδέν}
{
   return(i= =0);
}


int thetiko (int i)
/*Ελέγχει αν i > 0*/
{
   return( i > 0);
}
```

Ο ΑΤΔ Πραγματικός

Όπως για τον ΑΤΔ ακέραιο έτσι και για τον ΑΤΔ πραγματικό θα πρέπει να οριστούν το σύνολο των τιμών του και το σύνολο των βασικών πράξεών του. Το σύνολο των τιμών του είναι οι πραγματικοί αριθμοί ενώ ένα σύνολο βασικών πράξεών του είναι το εξής:

Καταχώρηση, Πρόσθεση, Αφαίρεση, Πολλαπλασιασμός, Διαίρεση, Μηδέν, Θετικός και Αποκοπή (ακέραιο τμήμα).



Μόνο μια περιοχή των πραγματικών αριθμών μπορεί να παρασταθεί λόγω του περιορισμού του μήκους της λέξης μνήμης.

Το συμπέρασμα είναι ότι και η υλοποίηση του ΑΤΔ πραγματικός δεν είναι τέλεια αφού υπάρχουν άπειροι πραγματικοί αριθμοί των οποίων η τιμή δεν μπορεί να παρασταθεί.

Ο ΑΤΔ Χαρακτήρας

Όπως για τους προηγούμενους ΑΤΔ, το σύνολο των χαρακτήρων ASCII είναι ένα υποσύνολο του συνόλου των χαρακτήρων του ΑΤΔ χαρακτήρας. Ενα σύνολο βασικών πράξεων είναι το ακόλουθο:

1. Καταχώρηση: Δέχεται ένα χαρακτήρα και τον καταχωρεί σε μια μεταβλητή.
2. Κωδικοποίηση: Δέχεται ένα χαρακτήρα και επιστρέφει έναν ακέραιο θετικό αριθμό.
3. Χαρακτήρας: Δέχεται έναν ακέραιο θετικό αριθμό και επιστρέφει ένα χαρακτήρα.

Ο ΑΤΔ Λογικός (Boolean)

Δύο είναι οι τιμές του ΑΤΔ Λογικός: true και false. Οι βασικές του πράξεις είναι : Καταχώρηση, Και (σύζευξη), Η (διάζευξη), Άρνηση.

Ο ΑΤΔ Πίνακας (Array)

Οι βασικές πράξεις του ΑΤΔ πίνακας είναι οι εξής:

1. Δημιουργία : Δημιουργεί ένα κενό πίνακα.

2. Καταχώρηση : Η πράξη αυτή ορίζει μια απεικόνιση από το σύνολο των δεικτών στο σύνολο των στοιχείων. Δέχεται σαν δεδομένα το όνομα του πίνακα, μια τιμή (ή τιμές) για το δείκτη (ή τους δείκτες) και ένα στοιχείο. Καταχωρεί το στοιχείο στη θέση του πίνακα που αντιστοιχεί στις τιμές των δεικτών.

3. Ανάκτηση: Η πράξη αυτή χρησιμοποιεί την παραπάνω απεικόνιση. Δέχεται σαν δεδομένα έναν πίνακα και ένα δείκτη (ή ένα σύνολο δεικτών) και επιστρέφει την πιο πρόσφατη τιμή που είχε αποθηκευτεί στη θέση που αντιστοιχεί στο δείκτη (δείκτες).

Υλοποίηση του ΑΤΔ Πίνακας

Για τη δημιουργία ενός πίνακα υποθέτουμε τους εξής ορισμούς και δηλώσεις:

```
#define  megethos ... /*αριθμός που ορίζεται από το χρήστη*/
typedef  ... /* ορίζεται από το χρήστη */ typos_stoixeiou;


typos_stoixeiou        a[megethos];    /*Δημιουργία*/
typos_stoixeiou        s;
```

```
void kataxorisi (typos_stoixeiou a[], int i, typos_stoixeiou s)
/*Καταχωρεί την τιμή του s στο a[i]}*/
{
        a[i]=s;
}


typos_stoixeiou anaktisi (typos_stoixeiou a[], int i)
/*Επιστρέφει την τιμή του a[i]*/
{
        return(a[i]);
}
```

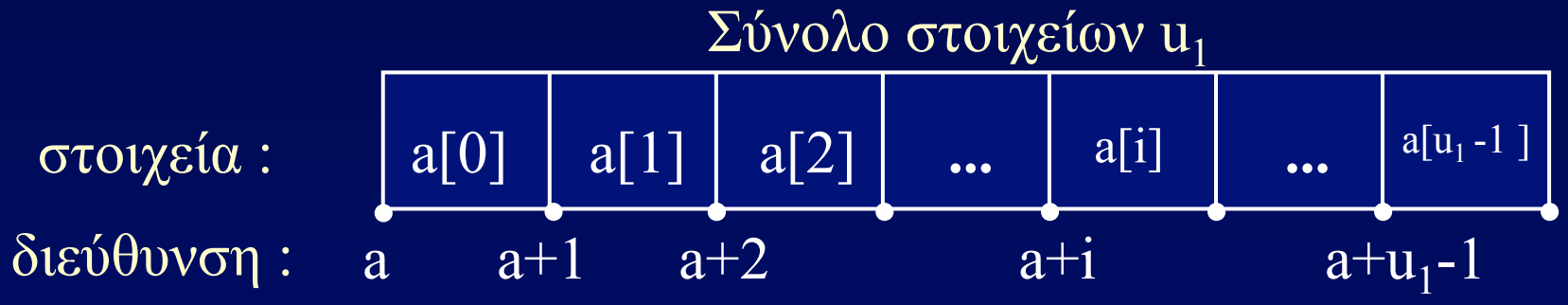
Μονοδιάστατοι Πίνακες

Στην C η δήλωση:

int a[u_1] ;

δεσμεύει ένα τμήμα από u_1 συνεχόμενες θέσεις μνήμης, κάθε μια από τις οποίες είναι αρκετά μεγάλη ώστε να περιέχει έναν ακέραιο.

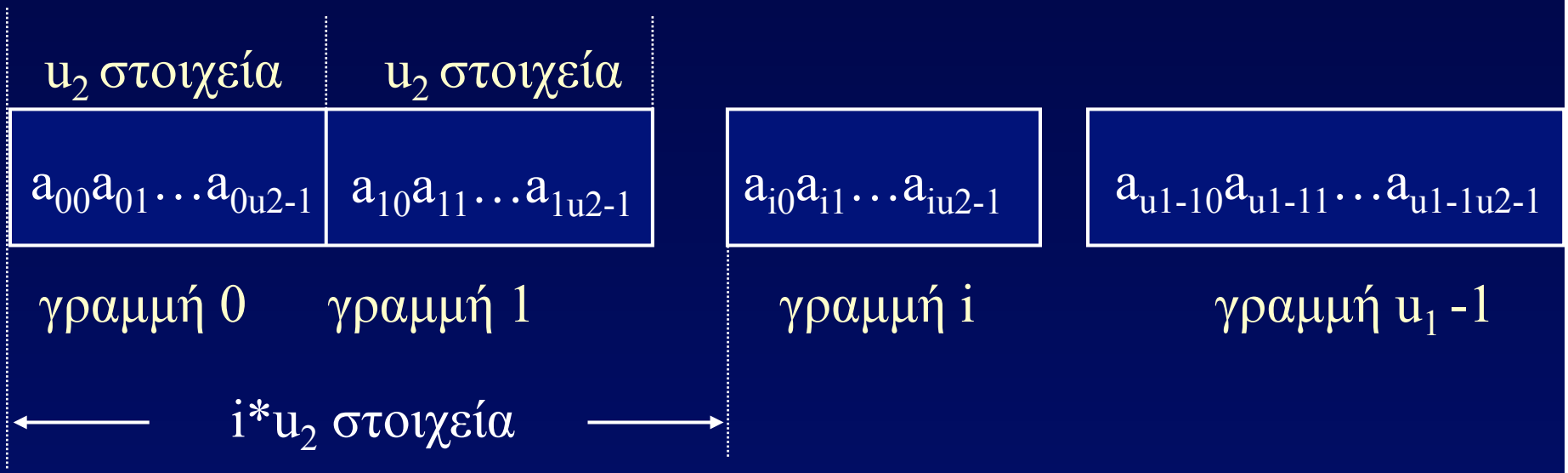
Ακολουθιακή παράσταση του a[0..u_1 -1]

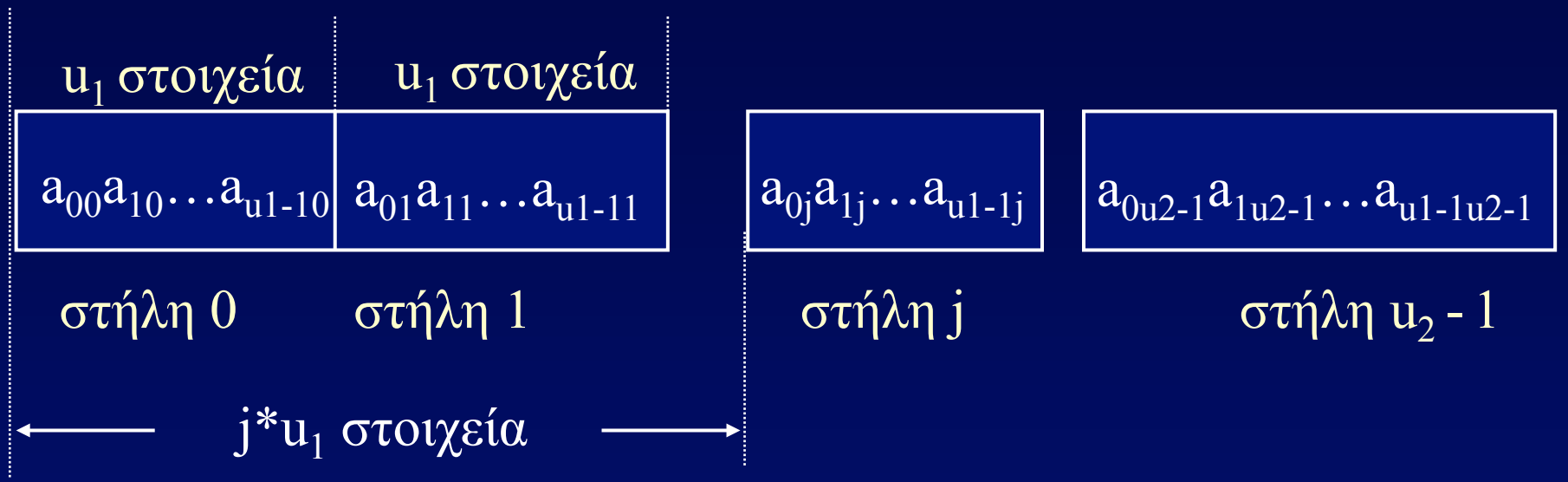


διεύθυνση (a[i]) = διεύθυνση (a[0]) + i = a + i

Ακολουθιακή παράσταση του a[u1][u2]

γραμμή1	a_{00}	a_{01}	a_{02}		a_{0u2-1}
γραμμή2	a_{10}	a_{11}	a_{12}		a_{1u2-1}
γραμμή3	a_{20}	a_{21}	a_{22}		a_{2u2-1}
•					
•					
•					
γραμμήu$_1$	a_{u1-10}	a_{u1-11}	a_{u1-12}		$a_{u1-1u2-1}$


u_2 στοιχεία
u_2 στοιχεία
$a_{00}a_{01}\ldots a_{0u2-1}$
$a_{10}a_{11}\ldots a_{1u2-1}$
$a_{i0}a_{i1}\ldots a_{iu2-1}$
$a_{u1-10}a_{u1-11}\ldots a_{u1-1u2-1}$
γραμμή 0
γραμμή 1
γραμμή i
γραμμή u_1 -1
i*u_2 στοιχεία


u_1 στοιχεία
u_1 στοιχεία
$a_{00}a_{10}\ldots a_{u1-10}$
$a_{01}a_{11}\ldots a_{u1-11}$
$a_{0j}a_{1j}\ldots a_{u1-1j}$
$a_{0u2-1}a_{1u2-1}\ldots a_{u1-1u2-1}$
στήλη 0
στήλη 1
στήλη j
στήλη u_2 - 1
$j*u_1$ στοιχεία

Αν θεωρήσουμε ότι ο πίνακας a είναι αποθηκευμένος ‘κατά γραμμές’ σε συνεχόμενες θέσεις μνήμης τότε:

Η σχέση : διεύθυνση (a[i][j]) = διεύθυνση (a[i][0]) + j

λόγω της σχέσης :

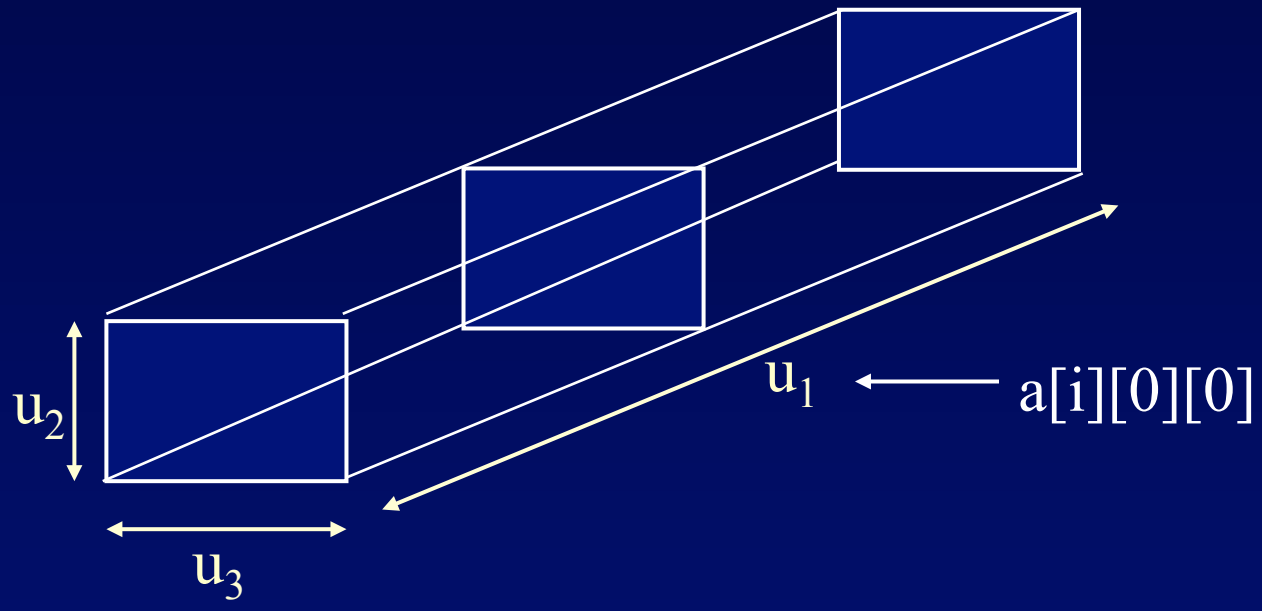
διεύθυνση (a[i] [0])= διεύθυνση (a[0][0]) + i*u2

γράφεται :

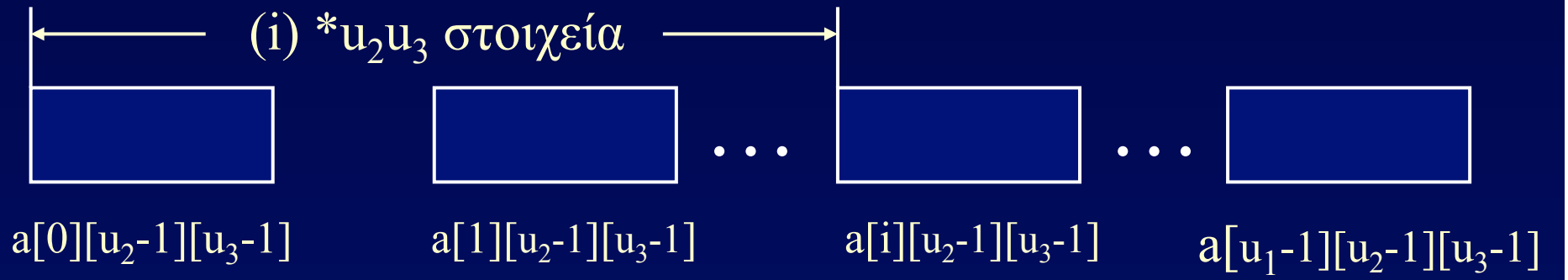
διεύθυνση (a[i][j]) = διεύθυνση (a[0][0]) + i*u2 + j

= βάση (a) + i*u2 + j

Θεώρηση του τρισδιάστατου πίνακα $a[u_1][u_2][u_3]$ σαν u_1 διδιάστατους πίνακες



Ακολουθιακή παράσταση ενός τρισδιάστατου πίνακα



Ο πίνακας αυτός μπορεί να θεωρηθεί ότι αποτελείται από ένα μονοδιάστατο πίνακα με u_1 στοιχεία, κάθε στοιχείο του οποίου είναι ένας δισδιάστατος πίνακας u_2 x u_3.

διεύθυνση (a[i, j, k]) = διεύθυνση (a[i, j, 0]) + k
= διεύθυνση (a[i, 0, 0]) + j*u_3 + k
= διεύθυνση (a[0, 0, 0]) + i*u_2 * u_3 + j*u_3 + k

Γενικά :

διεύθυνση (a[i_1, i_2, ... , i_n]) = διεύθυνση (a[0, 0, ... , 0]) +
i_1 * $u_2u_3...u_n$ + i_2 * $u_3u_4...u_n$ + i_3 * $u_4u_5...u_n$ + ...
+ i_{n-1} * u_n + i_n

$$= \beta\acute{\alpha}\sigma\eta(a) + \sum_{j=1}^{n} i_j \ \beta_j$$

$$\mu\varepsilon\ \beta_j = \prod_{k=j+1}^{n} u_k \ , 1 \le j \le n-1 \ , \quad \beta_n = 1$$

Ας σημειωθεί ότι το β_j μπορεί να υπολογισθεί από το β_{j+1} με ένα πολλαπλασιασμό αφού

$$\beta_n = 1,$$

$$\beta_{n-1} = u_n,$$

$$\beta_{n-2} = u_{n-1}\, u_n = u_{n-1}\, \beta_{n-1},$$

και γενικά

$$\beta_j = u_{j+1}\, u_{j+2} \ldots u_n = u_{j+1}\beta_{j+1}\ ,\ \ 1 <= j < n$$

Τ(διεύθυνση(a[i_1, i_2, …, i_n])) = O(n), αφού χρειάζονται:

- n-2 πολ/μοί για τον υπολογισμό των $\beta_n,\beta_{n-1},\beta_{n-2},\ldots,\beta_1$
- n-1 πολ/μοί και n προσθέσεις για τον υπολογισμό του $\sum_{j=1}^{n} i_j\ \beta_j$

Ειδικές μορφές πινάκων

Κάτω τριγωνικός πίνακας

$$\begin{bmatrix} a_{00} & 0 & 0 & \dots & 0 \\ a_{10} & a_{11} & 0 & \dots & 0 \\ a_{20} & a_{21} & a_{22} & \dots & 0 \\ . & . & . & & \\ . & . & . & & \\ . & . & . & & \\ a_{n-10} & a_{n-11} & a_{n-12} & \dots & a_{n-1n-1} \end{bmatrix}$$

Θα μπορούσαμε να τον αποθηκεύσουμε χωρίς τα μηδενικά για να κάνουμε οικονομία στη μνήμη.

Μπορούμε να αποθηκεύσουμε όλα τα μή-μηδενικά στοιχεία κατά γραμμές σε συνεχόμενες θέσεις μνήμης. Δηλαδή σε ένα μονοδιάστατο πίνακα b.

Ο πίνακας b θα έχει τα εξής στοιχεία:

a_{00}	a_{10}	a_{11}	a_{20}	a_{21}	a_{22}	. . .	a_{i0}	a_{i1}	. . .	a_{ii}	. . .	a_{n-10}	. . .	a_{n-1n-1}

$$\text{διεύθυνση}\,(a_{i,j}) = \text{διεύθυνση}\,(a_{i,0}) + j\ ,\ 0 \le j \le i \quad (1)$$

$$\text{Αλλά} \qquad \text{διεύθυνση}\,(a_{i,0}) = \sum_{k=1}^{i} k = \frac{i(i+1)}{2} \quad (2)$$

και η (1) σχέση δίνει

$$\text{διεύθυνση}\,(a_{i,j}) = \text{βάση}\,(a) + \frac{i(i+1)}{2} + j$$

Δηλαδή το στοιχείο a_{ij} έχει αποθηκευτεί στο στοιχείο

b[i(i+1)/2 + j]

a_{00}	$a_{10}a_{11}$	$a_{20}a_{21}a_{22}$	. . .	$a_{i0}a_{i1}\ldots a_{ii}$	. . .	$a_{n-10}a_{n-11}\ldots a_{n-1n-1}$

Αραιοί Πίνακες

Οι αραιοί πίνακες εμφανίζονται συχνά σε πολλές επιστημονικές εφαρμογές (μέθοδος Simplex, αριθμητική λύση μερικών διαφορικών εξισώσεων κ.α.).

$$\begin{bmatrix} 10 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 12 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -3 \\ 9 & 0 & 0 & -6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Αραιοί Πίνακες

Μπορούμε να αποθηκεύσουμε ένα δισδιάστατο αραιό πίνακα $a(u_1, u_2)$ σε ένα δισδιάστατο πίνακα $b(t+1,3)$, όπου t είναι ο αριθμός των μή-μηδενικών στοιχείων του πίνακα a.

Στην πρώτη γραμμή του πίνακα b (γραμμή 0) αποθηκεύουμε τους αριθμούς u_1, u_2 καθώς και των αριθμό t.

Στις υπόλοιπες t γραμμές του πίνακα b εργαζόμαστε ως εξής:

Αραιοί Πίνακες

Στα στοιχεία b[k,0] και b[k,1] αποθηκεύουμε τις τιμές των δεικτών i και j αντίστοιχα για τις οποίες το a_{ij} είναι το k-οστό μή-μηδενικό στοιχείο που συναντάμε στον πίνακα a, αν τον διατρέχουμε κατά γραμμές.

Στο στοιχείο b[k,2] αποθηκεύουμε την τιμή a_{ij} του k-οστού μή-μηδενικού στοιχείου.

Άσκηση: Δώστε έναν αλγόριθμο που να υπολογίζει σε ποια θέση του πίνακα b βρίσκεται αποθηκευμένο το στοιχείο a_{ij}. Ποιά είναι η πολυπλοκότητά χειρότερης περίπτωσης του αλγορίθμου σας;

Αποθήκευση του παραπάνω αραιού πίνακα κατά γραμμές

μή-μηδενικό στοιχείο	i	j	τιμή
	0	1	2
0	6	6	8
1	0	0	10
2	0	3	5
3	1	2	2
4	2	1	12
5	3	5	-3
6	4	0	9
7	4	3	-6
8	5	4	1

Ο ΑΤΔ εγγραφή (record)

Ο ΑΤΔ εγγραφή είναι μια πεπερασμένη συλλογή στοιχείων, που καλούνται πεδία (fields), γενικά διαφορετικού τύπου. Οι βασικές πράξεις του ΑΤΔ εγγραφή είναι:

1. Καταχώρηση: Δέχεται μια τιμή και την καταχωρεί σ' ένα πεδίο της εγγραφής.
2. Ανάκτηση: Επιστρέφει την τιμή ενός πεδίου της εγγραφής.

Υλοποίηση

typedef ... t_1 ;

typedef ... t_2 ;

.

.

typedef ... t_k ;

typedef struct {

t_1 p_1;

t_2 p_2;

.

.

.

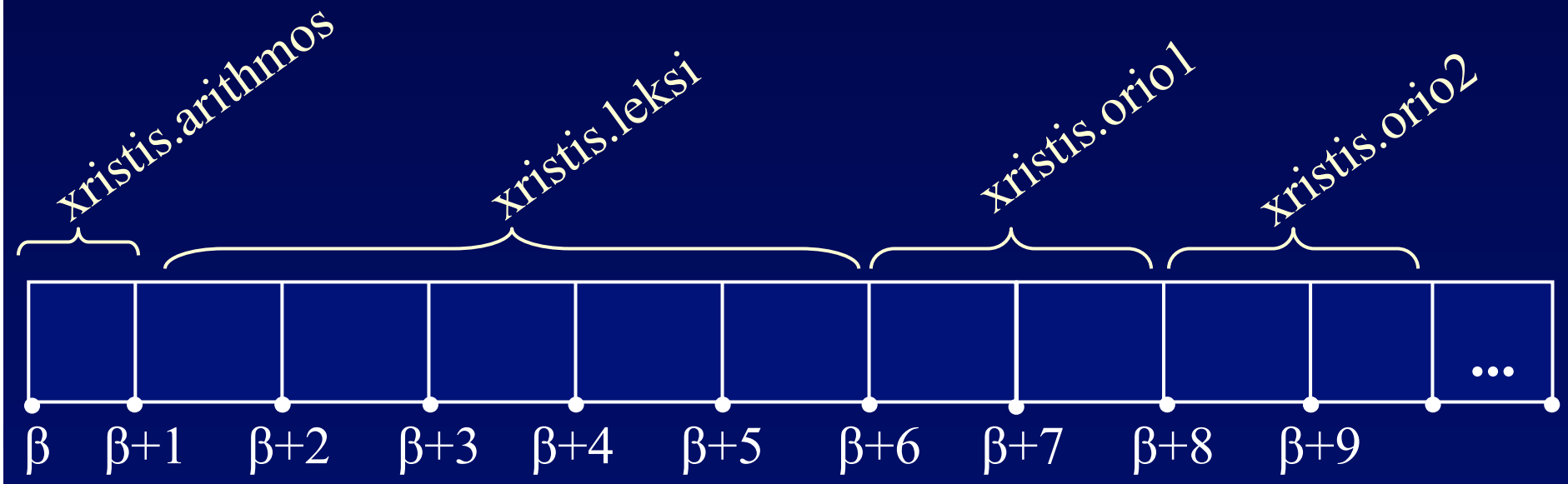
t_k p_k;

}typos_ergafis;

```
typedef struct{
        int       arithmos;
        char      leksi[10];
        float     orio1, orio2;
}ergafi;

ergafi   xristis;
```

Υλοποίηση Εγγραφής



Γενικά, για μια εγγραφή Ε με πεδία: Π_1 τύπου T_1, Π_2 τύπου T_2, ... , Π_v τύπου T_v, τα οποία απαιτούν λ_1, λ_2, ... , λ_v λέξεις μνήμης, αντίστοιχα, το πεδίο Ε . Π_i έχει διεύθυνση :

$$\text{διεύθυνση}\,(\text{E}\,.\,\Pi_i) =$$

$$= \text{διεύθυνση}\,(\text{E}\,.\,\Pi_1) + \lambda_1 + \lambda_2 + \ldots + \lambda_{i-1} =$$

$$= \text{βάση}\,(\text{E}) + \sum_{j=1}^{i-1} \lambda_j$$

όπου διεύθυνση (Ε.Π_i) είναι η διεύθυνση της πρώτης λέξης από ένα κομμάτι λ_i συνεχόμενων λέξεων.

Ο ΑΤΔ Σύνολο (set)

typedef enum{...} <name>

Παράδειγμα

typedef enum{sun=1,mon,tue,wed,thu,fri,sat} day

day mera;

Ο ΑΤΔ σύνολο αποτελείται από μια συλλογή μοναδικών στοιχείων του ίδιου τύπου. Οι πράξεις που συμπεριλαμβάνει ο ΑΤΔ σύνολο είναι οι ακόλουθες :

1. Δημιουργία

2. Δημιουργία Καθολικού συνόλου

3. Εισαγωγή στοιχείου

4. Διαγραφή στοιχείου

5. Μέλος

6. Κενό

7. Ισα

8. Υποσύνολο

9. Ενωση

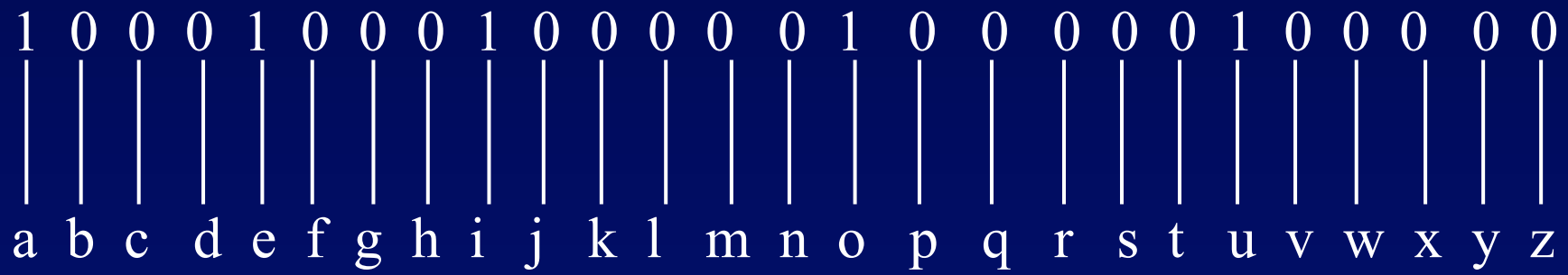
10. Τομή

11. Διαφορά

Υλοποίηση του ΑΤΔ Σύνολο με πίνακα

Παράσταση φωνηέντων

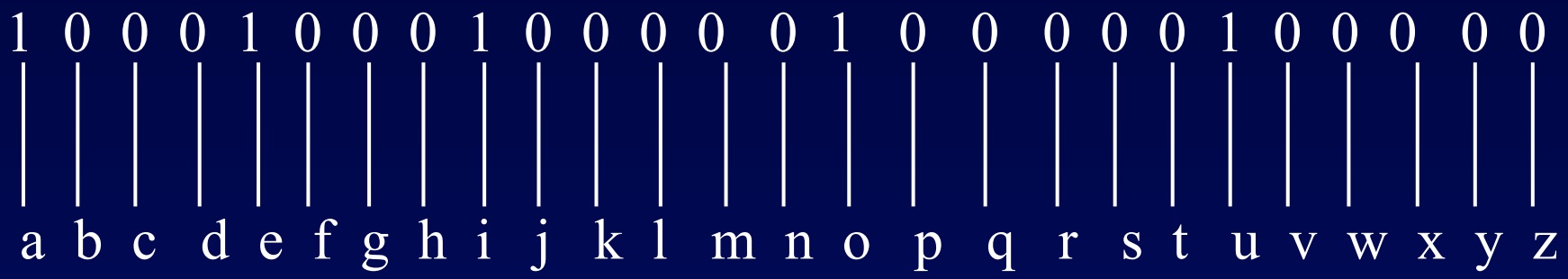
Παράδειγμα



Παράδειγμα

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
R:	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T:	1	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

$$T \cap R = \{a, c\} \leftrightarrow \text{and}$$



S[0]	S[1]	S[2]	S[3]	S[4]	S[5]	S[6]	S[7]	S[8]	S[9]
T	F	F	F	T	F	F	F	T	F

S[10]	S[11]	S[12]	S[13]	S[14]	S[15]	S[16]	S[17]	S[18]	S[19]
T	F	F	F	T	F	F	F	T	F

S[20]	S[21]	S[22]	S[23]	S[24]	S[25]
T	F	F	F	F	F

Όπου T = true
και F = false

Υλοποίηση του ΑΤΔ σύνολο με πίνακα

```
#define min_stoixeio  ...
#define max_stoixeio  ...
#define megisto_plithos     max_stoixeio - min_stoixeio+1

typedef int typos_synolou[megisto_plithos];
```

Στη συνέχεια παρουσιάζονται τα υποπρογράμματα που υλοποιούν τις βασικές πράξεις του ΑΤΔ σύνολο.

```
void dimiourgia(typos_synolou synolo)
/*Δημιουργεί ένα κενό σύνολο*/
{
        int i;
        for (i=0; i<= megisto_plithos - 1;i++)
                synolo[i]=0;
}
```

```
void katholiko(typos_synolou synolo)
{
        int i;
        for (i=min_stoixeio;i<=max_stoixeio;i++)
                eisagogi(i,synolo);
}
```

```
int eisagogi(int stoixeio, typos_synolou synolo)
{
        /* Εισάγει ένα στοιχείο στο σύνολο
        Επιστρέφει 1(true) αν πετύχει και 0 αλλιώς*/
        if ( (min_stoixeio<= stoixeio) && (
                stoixeio<=max_stoixeio) )
        {
                synolo[stoixeio-min_stoixeio]=1;
                return 1;
        }
        else
                return 0;
}
```

```
int diagrafi(int stoixeio, typos_synolou synolo)
/*Διαγράφει ένα στοιχείο από το σύνολο
Επιστρέφει 1(true) αν πετύχει και 0 αλλιώς */
{
        if ( (min_stoixeio<= stoixeio) && (stoixeio<=max_stoixeio) )
        {
                synolo[stoixeio-min_stoixeio]=0;
                return 1;
        }
        else
                return 0;
}
```

```
int melos(int stoixeio, typos_synolou synolo)
/*Ελέγχει αν ένα στοιχείο ανήκει στο σύνολο*/
{
        if ( (min_stoixeio<= stoixeio) &&
                (stoixeio<=max_stoixeio) )
        return synolo[stoixeio-min_stoixeio];
else
        return 0;
}
```

```
int keno_synolo(typos_synolou synolo)
/*Ελέγχει αν ένα σύνολο είναι κενό
Επιστρέφει 1 αν είναι κενό και 0 αλλιώς*/
{
        int i,keno;
        keno=1;
        i=min_stoixeio;
        while ((i<=max_stoixeio) && keno)
                if (melos(i,synolo))
                        keno=0;
                else
                        i++;
        return keno;
}
```

```
int isa_synola(typos_synolou s1, typos_synolou s2)
/*Ελέγχει αν τα δύο σύνολα s1 και s2 είναι ίσα*/
/*Αν είναι ίσα επιστρέφει 1 αλλιώς 0*/
{
        int i,isa;
        isa=1;
        i=min_stoixeio;
        while ((i<=max_stoixeio) && isa)
                if (melos(i,s1)!=melos(i,s2))
                        isa=0;
                else
                        i++;
        return isa;
}
```

```
int yposynolo(typos_synolou s1, typos_synolou s2)
/*Ελέγχει αν το s1 είναι υποσύνολο του s2*/
/*Αν ναι επιστρέφει 1 αλλιώς 0*/
{
        int i,yposnlo;
        yposnlo=1;
        i=min_stoixeio;
        while ((i<=max_stoixeio) && yposnlo)
                if (melos(i,s1) && !melos(i,s2))
                        yposnlo=0;
                else
                        i++;
        return yposnlo;
}
```

```
void enosi_synolou(typos_synolou s1,
                   typos_synolou s2,
                   typos_synolou  enosi)
/*Δημιουργεί την ένωση των συνόλων s1 και s2*/
{
        int i;
        for (i=min_stoixeio;i<=max_stoixeio;i++)
                enosi[i-min_stoixeio]=(melos(i,s1) || melos(i,s2));
}
```

```
void tomi_synolou(typos_synolou s1,
                  typos_synolou s2,
                  typos_synolou tomi)
/* Δημιουργεί την τομή των συνόλων s1 και s2*/
{
        int i;
        for (i=min_stoixeio;i<=max_stoixeio;i++)
                tomi[i-min_stoixeio]=(melos(i,s1) && melos(i,s2));
}
```

```
void diafora_synolou(typos_synolou s1,
                     typos_synolou s2,
                typos_synolou diafora)
/*Δημιουργεί τη διαφορά των συνόλων s1 και s2 (s1 - s2)*/
{
        int i;
        for (i=min_stoixeio;i<=max_stoixeio;i++)
        diafora[i-min_stoixeio]=(melos(i,s1) && !melos(i,s2));
}
```

Με την υλοποίηση αυτή, οι πράξεις της διαγραφής, της εισαγωγής, και της μέλος εκτελούνται σε σταθερό χρόνο, ενώ όλες οι άλλες πράξεις έχουν πολυπλοκότητα, O(n), όπου n είναι το πλήθος των στοιχείων του καθολικού συνόλου.

Το Κόσκινο του Ερατοσθένη (194 π.Χ.)

Η μέθοδος βρίσκει όλους τους πρώτους αριθμούς μεταξύ των ακεραίων 1, 2, 3, ..., n και είναι η ακόλουθη:

1. Δημιουργείται το σύνολο κόσκινο που περιέχει τους ακεραίους από το 1 μέχρι το n.
2. Διαγράφεται ο 1 γιατί, σύμφωνα με τον ορισμό των πρώτων αριθμών, το 1 δεν είναι πρώτος. Στη συνέχεια το μικρότερο στοιχείο που απομένει στο κόσκινο είναι ο επόμενος πρώτος αριθμός.
3. Οσο το κόσκινο δεν είναι κενό να εκτελούνται:
   Να βρεθεί το μικρότερο στοιχείο στο κόσκινο.
   Να διαγραφτεί το μικρότερο στοιχείο και τα πολλαπλασιά του (τα οποία δεν είναι πρώτοι) από το κόσκινο.

Το Κόσκινο του Ερατοσθένη

Η υλοποίηση του παραπάνω αλγορίθμου με τη χρήση συνόλων είναι η ακόλουθη:

```
void koskino_Eratostheni(void)
{
   typos_synolou        koskino;
   int                  epomeno /*ο επόμενος πρώτος αριθμός*/
   int                  j;

   katholiko (koskino);
   diagrafi (1, koskino);
   epomeno = 1;
```

συνέχεια

```
do
{
/*Το μικρότερο στοιχείο στο κόσκινο είναι ο επόμενος πρώτος αριθμός*/
     while (!melos (epomeno, koskino))
                    epomeno++ ;
     printf ("%d",epomeno);
     /*Διαγραφή του epomeno και των πολλαπλασίων του από το koskino*/
     j= epomeno;
     while   (j <= megisto_plithos)
     {
        diagrafi (j, koskino);
        j = j+epomeno;
     }
}while (!keno_synolo (koskino));
} /*koskino_Eratostheni*/
```