ΗΥ-150
Προγραμματισμός

Πίνακες (Arrays)

Εισαγωγικά

- Έστω ότι θέλουμε να αποθηκεύσουμε 100 ονόματα φοιτητών και τους βαθμούς τους. Πως θα το κάναμε αυτό με μεταβλητές;
- Πως θα μπορούσαμε να πούμε με αυτό το τρόπο «ταξινόμησε τους βαθμούς και τύπωσε τους»;
- Πως θα μπορούσαμε να πούμε «τύπωσε μόνο εκείνους που πέρασαν το μάθημα»;
- ΛΥΣΗ: Πίνακες
  - Συλλογή μεταβλητών ίδιου τύπου
  - Σταθερό μέγεθος
  - Ν στοιχεία: 0 … Ν-1
  - **Στη C η αρίθμηση ξεκινάει από το 0**

Πίνακες

- Ένα σύνολο διατεταγμένων (συνήθως σχετιζόμενων) πραγματικών αριθμών
  - π.χ. που καταγράφουν τη θερμοκρασία το μεσημέρι σε μία πόλη κατά τη διάρκεια του έτους
  - αποθηκεύονται σε ένα μονοδιάστατο πίνακα (array) πραγματικών αριθμών με 365 στοιχεία παρά σε ισάριθμες ανεξάρτητες μεταβλητές.
- *Η χρήση πίνακα διευκολύνει πολύ την αναζήτηση, ταξινόμηση και, γενικά, διαχείριση σχετιζόμενων ποσοτήτων.*
- Η δήλωση ενός μονοδιάστατου πίνακα γίνεται ως εξής:

  τύπος όνομα[πλήθος_στοιχείων];

- Το πλήθος_στοιχείων πρέπει να είναι μία ακέραιη σταθερά ή μία ακέραιη σταθερή ποσότητα, γνωστή κατά τη μεταγλώττιση του κώδικα.

Παράδειγμα

- Οι παρακάτω δηλώσεις ορίζουν τον πραγματικό πίνακα temperature με 365 στοιχεία και ένα πίνακα των (τετραψήφιων) εσωτερικών τηλεφώνων μιας εταιρίας (που χωρούν σε **int**):

double temperature[365];

int const N = 155;

int telephone[N];

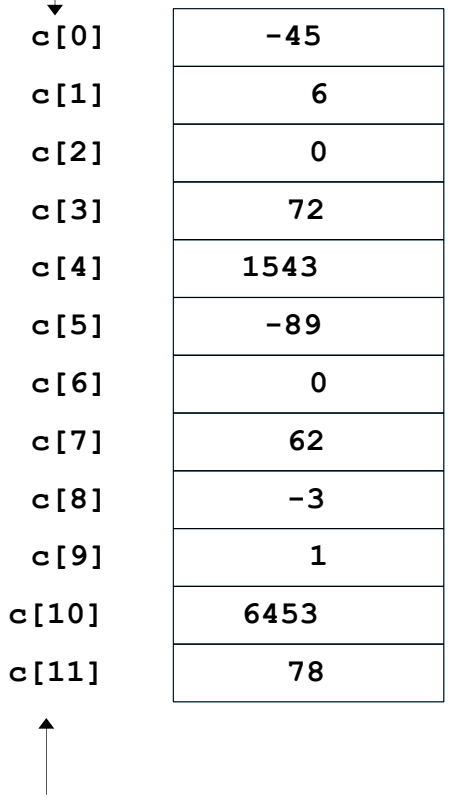
Η δήλωση του telephone έγινε με "παράμετρο" το πλήθος των στοιχείων ώστε μία αλλαγή του να μην απαιτεί εκτεταμένες τροποποιήσεις στον κώδικα.

Πίνακες

- Πίνακας
  - Σύνολο από συνεχόμενες θέσεις μνήμης
  - Ίδιο όνομα και ίδιος τύπος
  - Ορισμός : int c[12];
- Για να πάρουμε ένα στοιχείο του πίνακα δίνουμε:
  - Όνομα πίνακα
  - Θέση
  - c[1] = 6;
- Μορφή:

  *arrayname***[** *position number* **]**
  - Πρώτο στοιχείο στη θέση 0
  - Για έναν πίνακα **n** στοιχείων με όνομα **c**:
    - **c[0]**, **c[1]**, ..., **c[n – 1]**

Όλα τα στοιχεία του πίνακα έχουν όνομα `c`



Θέση μέσα στον πίνακα **`c`**

- int c[12];
- c[12] = 0;
- Γιατί μετράμε από το 0;

Παράδειγμα

- Πρόσβαση στα στοιχεία ενός πίνακα, π.χ. του temperature, γίνεται βάζοντας σε αγκύλες μετά το όνομα του πίνακα ένα ακέραιο μεταξύ 0 και D-1 όπου D η διάσταση.
- Το πρώτο, δηλαδή, στοιχείο του πίνακα είναι στη θέση 0. Π.χ.
- temperature[0] = 14.0;
- temperature[1] = 14.5;
- temperature[2] = 15.5;
- temperature[3] = 13.0;
- temperature[4] = 15.0;

```
for (i=0;i<N;i++)
{
   int foo;
   cin >> foo;
   temperature[i] = foo;

   cout << temperature[i];
}
```

Πίνακες

- Δήλωση πολλών πινάκων ίδιου τύπου
  - Όπως και μεταβλητές
  - **int b[100], x[27];**
- Τα στοιχεία του πίνακα χρησιμοποιούνται σαν συνηθισμένες μεταβλητές

  c[0] = 3;

  cout << c[0];
- Μπορούμε να κάνουμε πράξεις μέσα στο δείκτη.
  - if (**x[0]** == **3)**
  - **c[5 - 2] == c[3] == c[i]**
- Εμφωλιασμένοι δείκτες: c[c[c[1]]]
- ΠΡΟΣΟΧΗ:
  - ***δείκτης πίνακα*** (index) c[i], index is i (int i)

Στοιχεία του Πίνακα σε Εκφράσεις

- έκφραση1 ← όνομα_πίνακα[έκφραση2]
  - πρώτα υπολογίζεται η τιμή της έκφρασης2
  - η τιμή της έκφρασης2 πρέπει να είναι ακέραια (int)
  - η τιμή της έκφρασης1 είναι το στοιχείο του πίνακα που αντιστοιχεί στην συγκεκριμένη θέση που δίνεται από την έκφραση2
  - ο τύπος της έκφρασης1 είναι ο τύπος των στοιχείων του πίνακα

```
double array[100];
int i, x=5;

for (i = 0; i < 100;  ++i)
   cin >> array[i];

i = (int)(array[10*x]);
```

Πρόγραμμα - armemory.c

```
#include <iostream.h>
#define SIZE 10

int main()
{
    int array[SIZE] = {5, 3,9, 123,0, 293,   2394, 2,-1, -232};

    int n;

    for (n=0; n < SIZE ; n++)
    {
        cout << "Element array[" << n << "] = ";
        cout << array[n] << “, memory address ";
        cout << &array[n];
    }

    return 0;
}
```

Δήλωση πινάκων

- Όταν δηλώνουμε πίνακες ως μεταβλητές καθορίζουμε:
  - Όνομα
  - Τύπο
  - Αριθμό στοιχείων

    arrayType arrayName[numberOfElements];

    int c[10];

    float myArray[3284];
- Όταν δηλώνουμε πίνακες ως ορίσματα στον ορισμό συναρτήσεων καθορίζουμε:
  - Όνομα
  - Τύπο
  - **int myfunction(int c[]){ ... }**
- Όταν δηλώνουμε πίνακες ως ορίσματα στη δήλωση συναρτήσεων καθορίζουμε:
  - **Τύπο**
  - **int myfunction(int []);**

Παραδείγματα χρήσης

- Αρχικοποίηση

  int n[5] = { 1, 2, 3, 4, 5 };

  – Αν δεν υπάρχουν αρκετά στοιχεία για αρχικοποίηση, τότε τα υπολοιπόμενα γίνονται 0

  ```
  int n[ 5 ] = { 0 };   /* All elements 0 */
  for (i = 0; i < 5; ++i)
      n[i] = 0;
  ```

  – Αν βάλουμε πιο πολλά στοιχεία από όσα έχει ο πίνακας τότε θα πάρουμε λάθος

- Αν αρχικοποιούμε τον πίνακα μπορούμε να μη δώσουμε μέγεθος και ο πίνακας θα είναι όσος και τα στοιχεία που δίνουμε

  int n[ ] = { 1, 2, 3, 4, 5 };

  – 5 στοιχεία, οπότε πίνακας 5 στοιχείων

Αρχικοποίηση

- Μπορούμε να δώσουμε αρχικές τιμές στα στοιχεία ενός πίνακα εάν κατά τον ορισμό του παραθέσουμε λίστα τιμών με ίδιο (ή μετατρέψιμο) τύπο με τα στοιχεία.
- Κατά τη δήλωση με αρχική τιμή μπορούμε να παραλείψουμε τη διάσταση του πίνακα οπότε υπολογίζεται από τον compiler με βάση το πλήθος των παρατιθέμενων τιμών.
- Αν υπάρχει και η διάσταση δεν επιτρέπεται να δίνονται περισσότερες αρχικές τιμές, ενώ, αν παρατίθενται λιγότερες, οι υπόλοιπες θεωρούνται 0 (μετατρεπόμενο στον αντίστοιχο τύπο):
- **int** primes[5] = **{**1,2,3,5,7**}**;
- **int** digits[] = **{**0,1,2,3,4,5,6,7,8,9**}**; // size is 10
- **char** alphabet[5] = **{**'a', 'b', 'c', 'd', 'e', 'f'**}**; // Error
- **int** a[5] = **{**12, 5, 4**}**; // a == {12,5,4,0,0}

Όρια

Προσέξτε ότι αν δώσετε δείκτη (index) εκτός των ορίων του πίνακα, δηλαδή κάτω από το 0 ή πάνω από D-1 δε θα διαγνωστεί ως λάθος από τον compiler.

Στη C++ δεν υπάρχει έλεγχος ορίων!

Τι γίνεται αν βγούμε έξω από τον πίνακα;

Προγράμματα (outofbounds.c)

```
#include <stdio.h>

int main()
{
    int c,b, n[10], a;

    a = 5;
    b = 6;
    cout <<"a=" << a << " b="<<b<<endl;

    // Here we get out of bounds. What's going to happen depends on the system.
    for (c=-3; c < 10; c++)
            n[c] = 10;
    n[10] = 0;

    cout <<"a=" << a << " b="<<b<<endl;

    return 0;
}
```

Τι Κρύβει το Όνομα Ενός Πίνακα

- int a[100];
- a[0] το 1ο στοιχείο του πίνακα
- &a[0] η διεύθυνση του 1ου στοιχείου του πίνακα
- &a[0] ισοδύναμο με το a (το όνομα του πίνακα)
  - Αντί να γράφουμε &a[0] απλά γράφουμε a
  - Άρα το a είναι διεύθυνση (δείκτης) στο πρώτο στοιχείο
  - Ο τύπος του a είναι σταθερός δείκτης σε ακέραιο
- Μπορούμε να χρησιμοποιούμε το a όπως ένα δείκτη
  - αργότερα θα δούμε ότι μπορούμε να χρησιμοποιούμε και δείκτες ως πίνακες!
- Το όνομα του πίνακα αντιστοιχεί σε μια **σταθερή διεύθυνση**

Πίνακες σαν ορίσματα συναρτήσεων

- Ορίσματα: Δίνουμε το όνομα του πίνακα μόνο:

```
int myArray[ 24 ];
float getMean(int myArray[],int length)
{
        int i;
        float sum = 0;

        for (i = 0; i < length; i++)
                sum += myArray[i];

        return sum/length;
}
getMean( myArray, 24 );
```

  - ***Συνήθως*** περνάμε μαζί και το μέγεθος του πίνακα
  - **Οι πίνακες πέρνιουνται call-by-reference**
  - Το όνομα του πίνακα είναι και δείκτης στο 1ο στοιχείο
  - Η συνάρτηση έχει πρόσβαση στο σημείο αποθήκευσης του πίνακα
- Περνάμε στοιχεία του πίνακα
  - call-by-value
  - Περνάμε όνομα[θέση]: **myArray[3]**

- Πως μπορεί κανείς να αποφύγει να περάσει τον αριθμό στοιχείων του πίνακα;

Από τη μνήμη που δεσμεύουν:

- το μέγεθος όλου του πίνακα / μέγεθος στοιχείου
- *κατάλληλη χρήση sizeof() για την εύρεση των τιμών τους*

Size-of

- **sizeof** μπορεί να χρησιμοποιηθεί με
  - Ονόματα μεταβλητών
  - Ονόματα τύπων
  - Σταθερές

- **sizeof**
  - Επιστρέφει το μέγεθος σε bytes
  - Για πίνακες: μέγεθος του 1 στοιχείου * πλήθος
  - Αν **sizeof(int)** είναι **4** bytes, τότε

```
int myArray[ 10 ];
cout << sizeof( myArray );
```

Θα τυπώσει **40**

Υπολογισμός Διεύθυνσης Στοιχείων

- Πού βρίσκεται στη μνήμη το στοιχείο a[index];
  - **double a[100];**
  - Έστω ο πίνακας αρχίσει στην διεύθυνση a
  - Κάθε στοιχείο του πιάνει χώρο sizeof(double)
  - Θέλουμε να προσπελάσουμε το στοιχείο με δείκτη index
  - Η διεύθυνση (σε Bytes) στη μνήμη είναι η …

    a + sizeof(double) * index

Πρόγραμμα : histogram.cpp

```
#include <iostream.h>
// The size of the array storing the grades
#define HIST_SIZE 11
#define STUDENTS 30
// The size of the array storing the
    histogram

//Computes the histogram n of array a.
//Array a has sizeA elements
void computeHist(int n[],int a[],int sizeA)
{
    int i;

    for (i = 0; i < sizeA; ++i)
        n[a[i]]++;
}
```

```
int main()
{
    int hist[HIST_SIZE] = {0};
    int ba8moi[STUDENTS],i,j;

    for (i = 0; i < STUDENTS; ++i)
        cin >> ba8moi[i];

    computeHist(hist,ba8moi,STUDENTS);

    for (i=0; i < HIST_SIZE; i++)
    {
        cout << i << "-" << hist[i] << ' ';

        for (j = 0; j < hist[i]; j++)
            cout << '*';

        cout << endl;
    }

    return 0;
}
```

Πρόγραμμα

```
#include <stdio.h>

void f(int [], int size, int b);

int main()
{
    int c[10] = {0};
    int n;
    int q = 30;

    /* Print the values of the array
    before the function call */
    for (n=0; n < 10; n++)
        cout << n << " " << c[n];

    cout << "\nq = " << q << "\n\n";

    f(c, 10, q);
```

```
/* Print the values of the array AFTER
    the function call
   Notice: the values of the array
    elements have changed.
    The value of q has not changed.*/

    for (n=0; n < 10; n++)
        cout << n << " " << c[n];

    cout << "\nq = " << q << "\n\n";

    return 0;
}

void f(int a[], int size, int b)
{
    for (int i=0; i<size; i++)
        a[i] = 5;

    b = 100;
}
```

ΗΥ-150
Προγραμματισμός

Ταξινόμηση και Αναζήτηση

Το πρόβλημα της Αναζήτησης

- Δοθέντος δεδομένων, λ.χ. σε Πίνακα (P)
- Ψάχνω να βρω κάποιο συγκεκριμένο στοιχείο (key)
- Αν ο πίνακας δεν είναι ταξινομημένος
  - Γραμμική Αναζήτηση (Linear search)
    - Απλούστερη δυνατή
    - Σύγκρινε σειριακά κάθε στοιχείο του πίνακα με την τιμή-κλειδί
    - Χρήσιμο για μικρούς και ΜΗ ταξινομημένους πίνακες

```
int linearSearch(int P[],int apo,int eos,int key)
{
   int i;

   for (i = apo; i <= eos; ++i)
   {
        if (P[i] == key)
            return i;
   }
   return -1;
}
```

Το πρόβλημα της Αναζήτησης

- Αν ο πίνακας είναι ταξινομημένος
  - λ.χ. τηλεφωνικός κατάλογος
  - Μπορώ να κάνω πολύ πιο γρήγορα την αναζήτηση
- Δυαδική Αναζήτηση (Binary Search)
  - Συγκρίνει το P[**`middle`**] στοιχείο με το ζητούμενο **`key`**
    - Αν είναι ίσα βρέθηκε
    - Αν **`key < P[middle]`**, ψάχνει στο 1ο μισό του πίνακα
    - Αν **`key > P[middle]`**, ψάχνει στο 2ο μισό του πίνακα
    - Επανάληψη
  - Πολύ γρήγορη – χειρότερη περίπτωση $\log_2(N)$, N # στοιχείων πίνακα
    - Πίνακας 100 στοιχείων χρειάζεται το πολύ 7 βήματα
    - Πίνακας 100.000 στοιχείων χρειάζεται το πολύ 20 βήματα
    - Πίνακας 100.000.000 στοιχείων χρειάζεται το πολύ 27 βήματα

Binary Search – Υλοποίηση με επανάληψη

```
int binaryLoopSearch(int p[], int searchkey, int low, int high)
{
   int middle;
   while ( low <= high )
   {
        middle = (low + high ) / 2;

        if (searchkey == p[middle])
               return middle;
        else if (searchkey < p[middle] )
               high = middle - 1;
        else
               low = middle + 1;
   }
   return -1;
}
```

Binary Search – Υλοποίηση με αναδρομή

```
int binarySearch(int p[], int searchkey, int low, int high)
{
   int middle;

   middle = (low + high ) / 2;
    if (high < low)
        return -1;
    if (searchkey == p[middle])
        return middle;
   else if (searchkey < p[middle])
        return binarySearch(p, searchkey, low, middle-1);
   else
        return binarySearch(p, searchkey, middle+1,high);

   return -1;
}
```

Το πρόβλημα της Ταξινόμησης

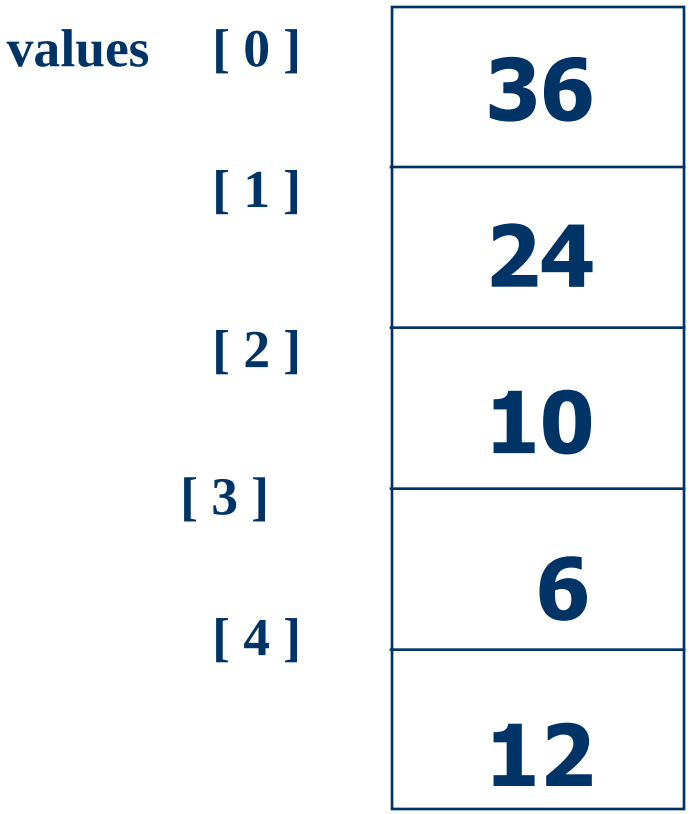
- Δοθέντος δεδομένων, λ.χ. σε Πίνακα (P)

- Να γίνει αναδιάταξη των στοιχείων του ώστε να βρεθούν σε αύξουσα (ή φθίνουσα) σειρά, δηλαδή ταξινομημένα

12	31	4	32	134	13	42	1	43	2
1	2	4	12	13	31	32	42	43	134

- Λύση : Υπάρχουν διάφοροι αλγόριθμοι που το πετυχαίνουν και διαφοροποιούνται στο υπολογιστικό
  - Straight Selection Sort - Bubble Sort : $O(N^2)$
  - Quick Sort : O(N log N)

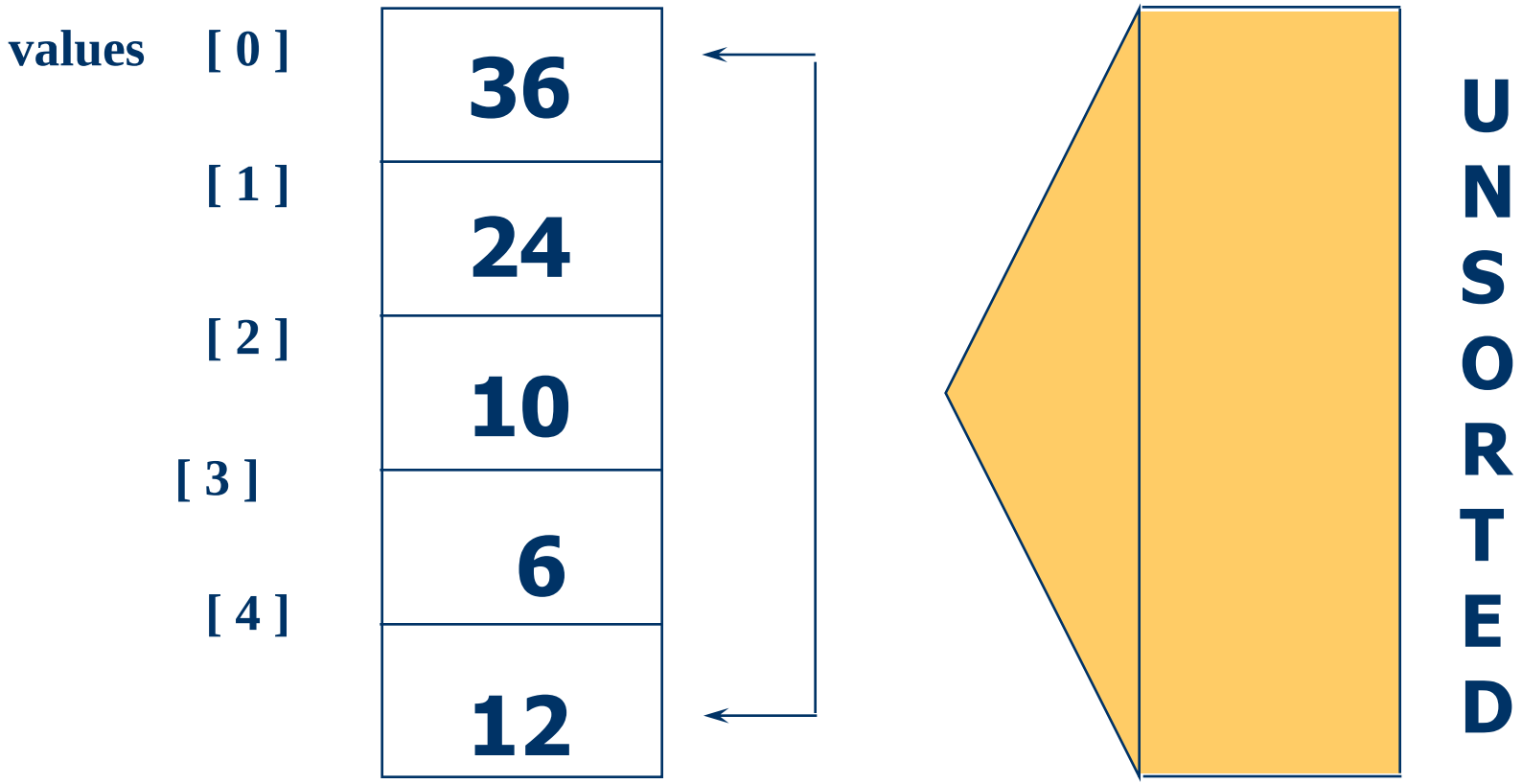
Ταξινόμηση με επιλογή



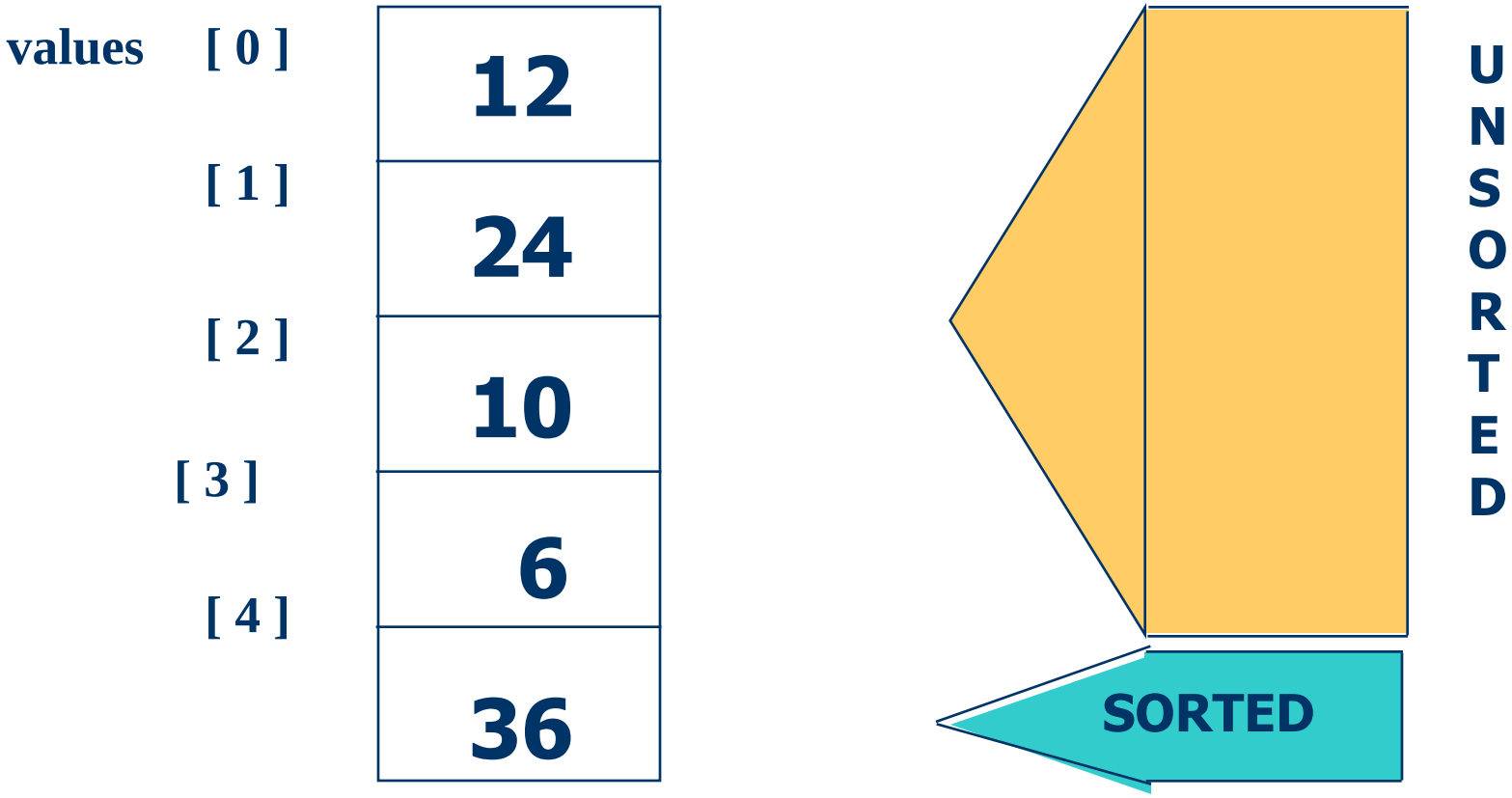
Χωρίζει νοητά τον πίνακα σε 2 μέρη:
-Ταξινομημένο
-Μη ταξινομημένο

Σε κάθε επανάληψη βρίσκει το μεγαλύτερο στοιχείο και το ανταλλάσει με το πρώτο του αταξινόμητου υποπίνακα.

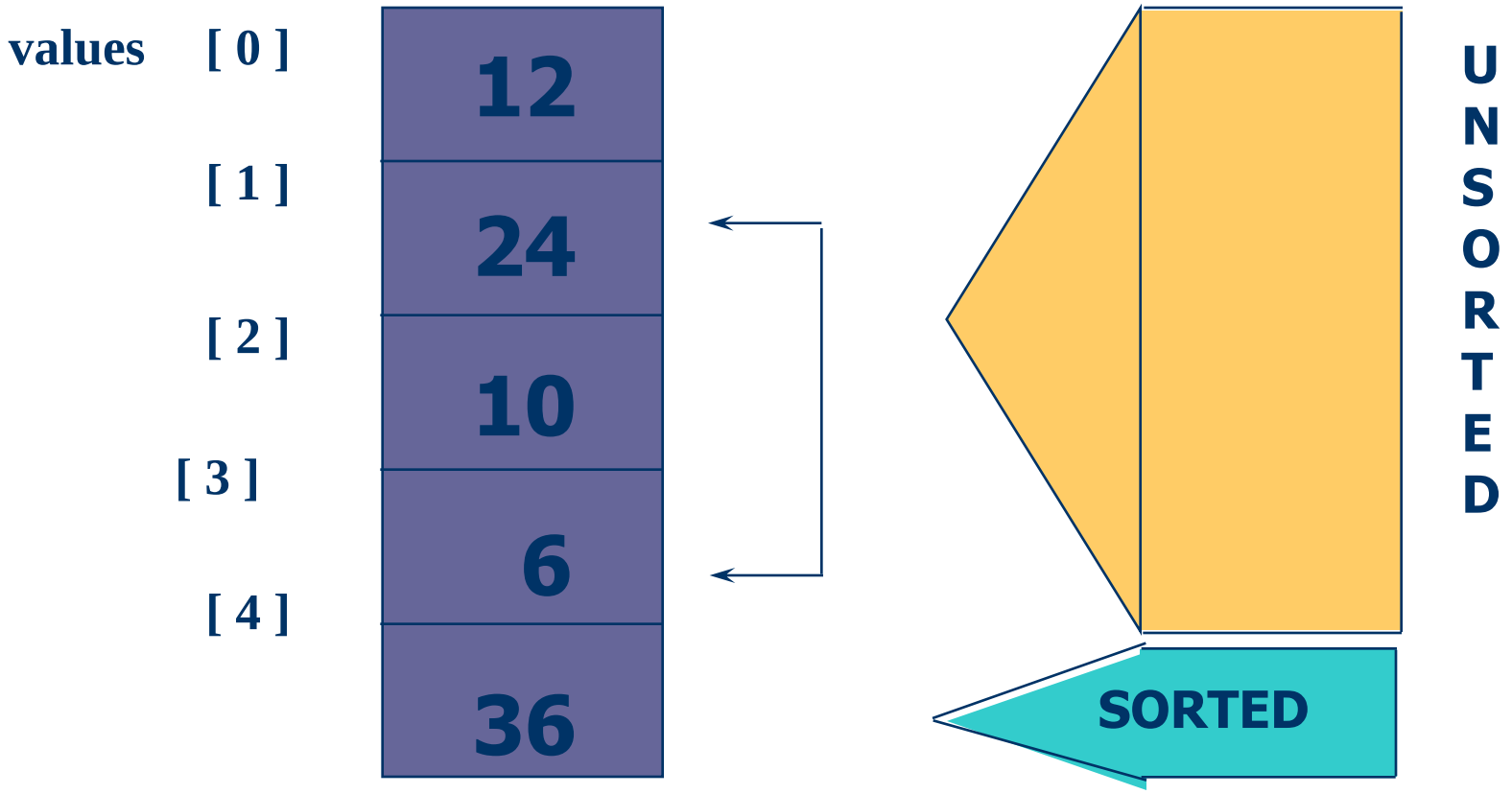
Ταξινόμηση με επιλογή



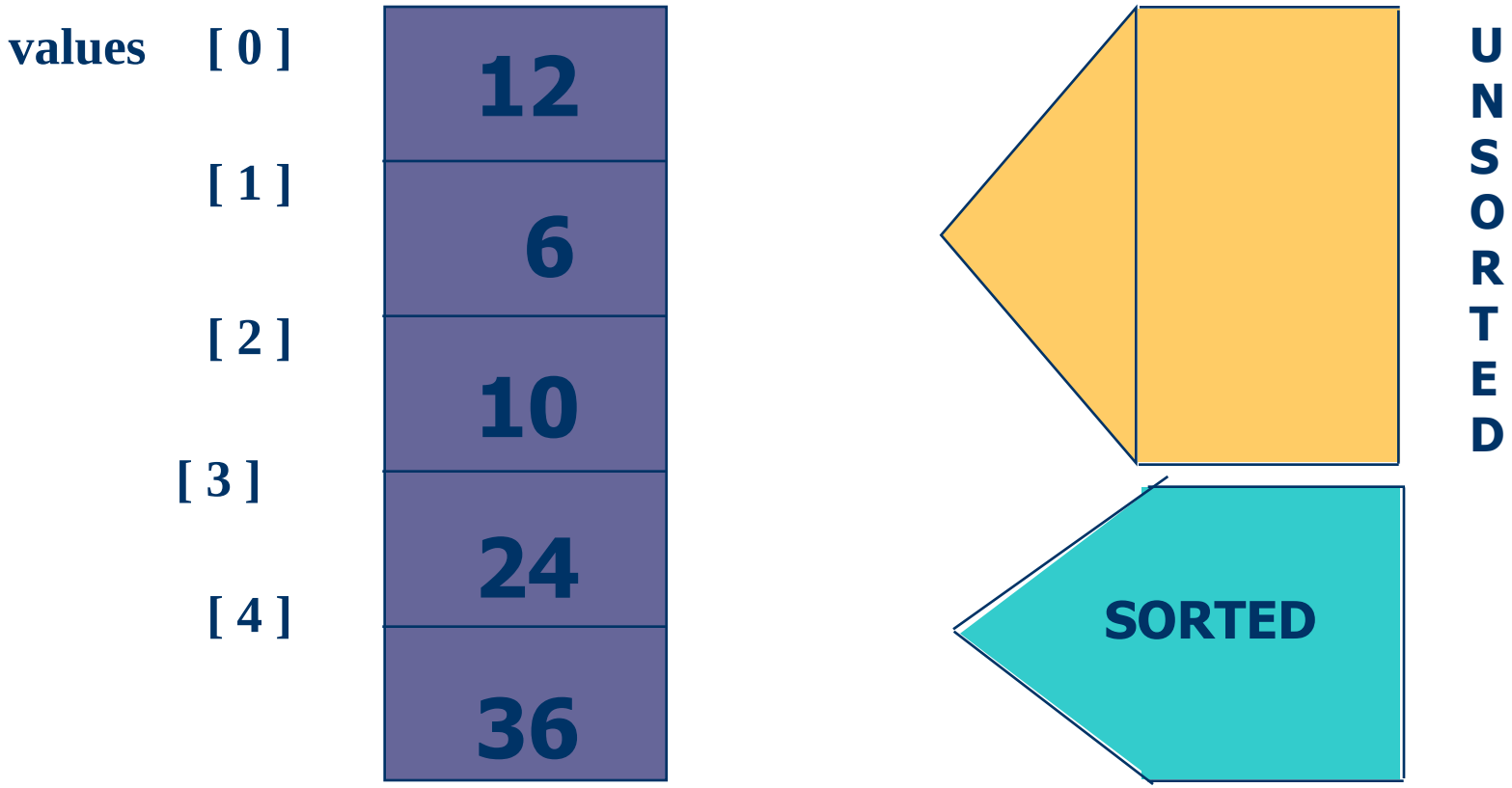
Ταξινόμηση με επιλογή



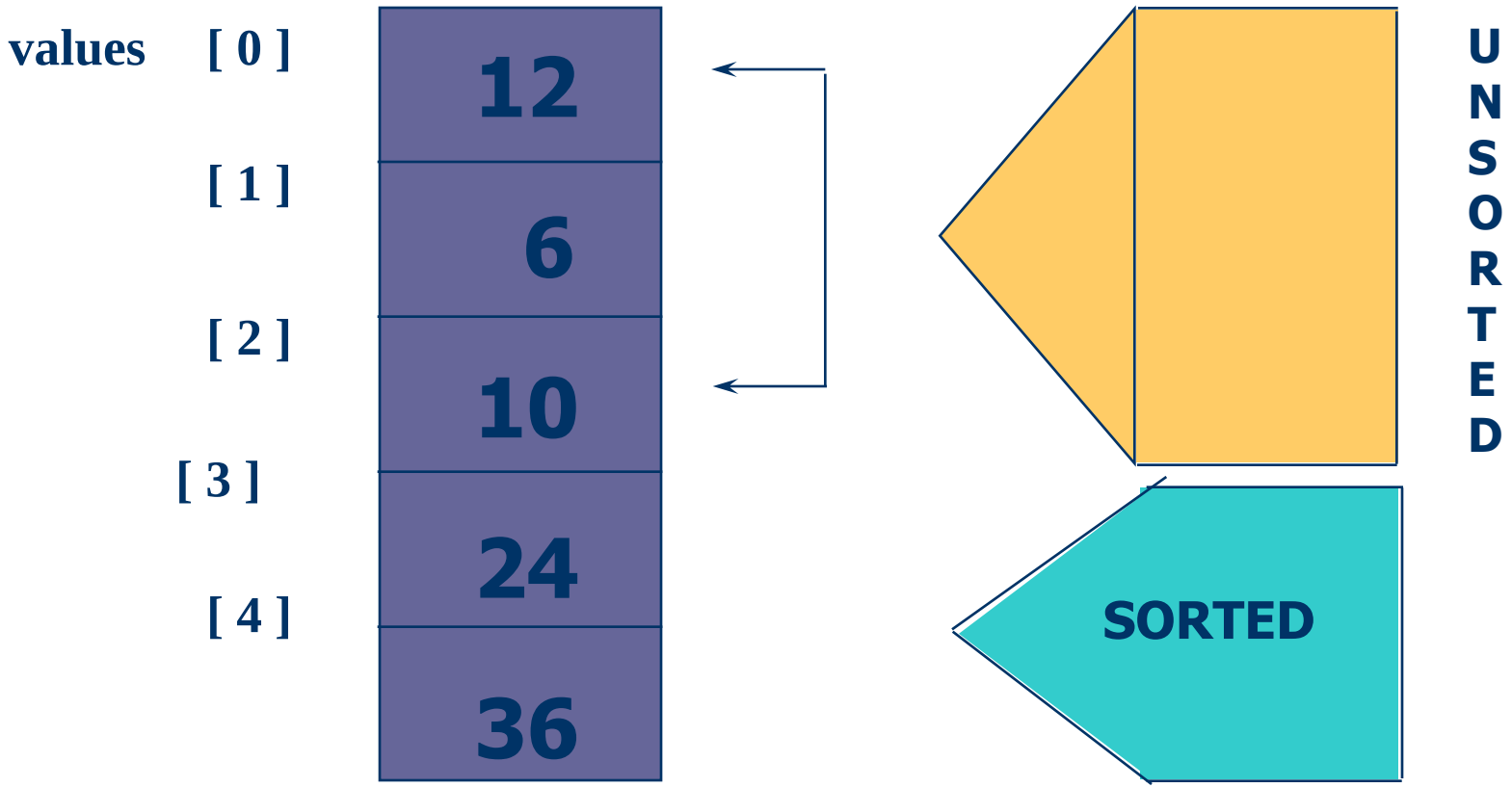
Ταξινόμηση με επιλογή



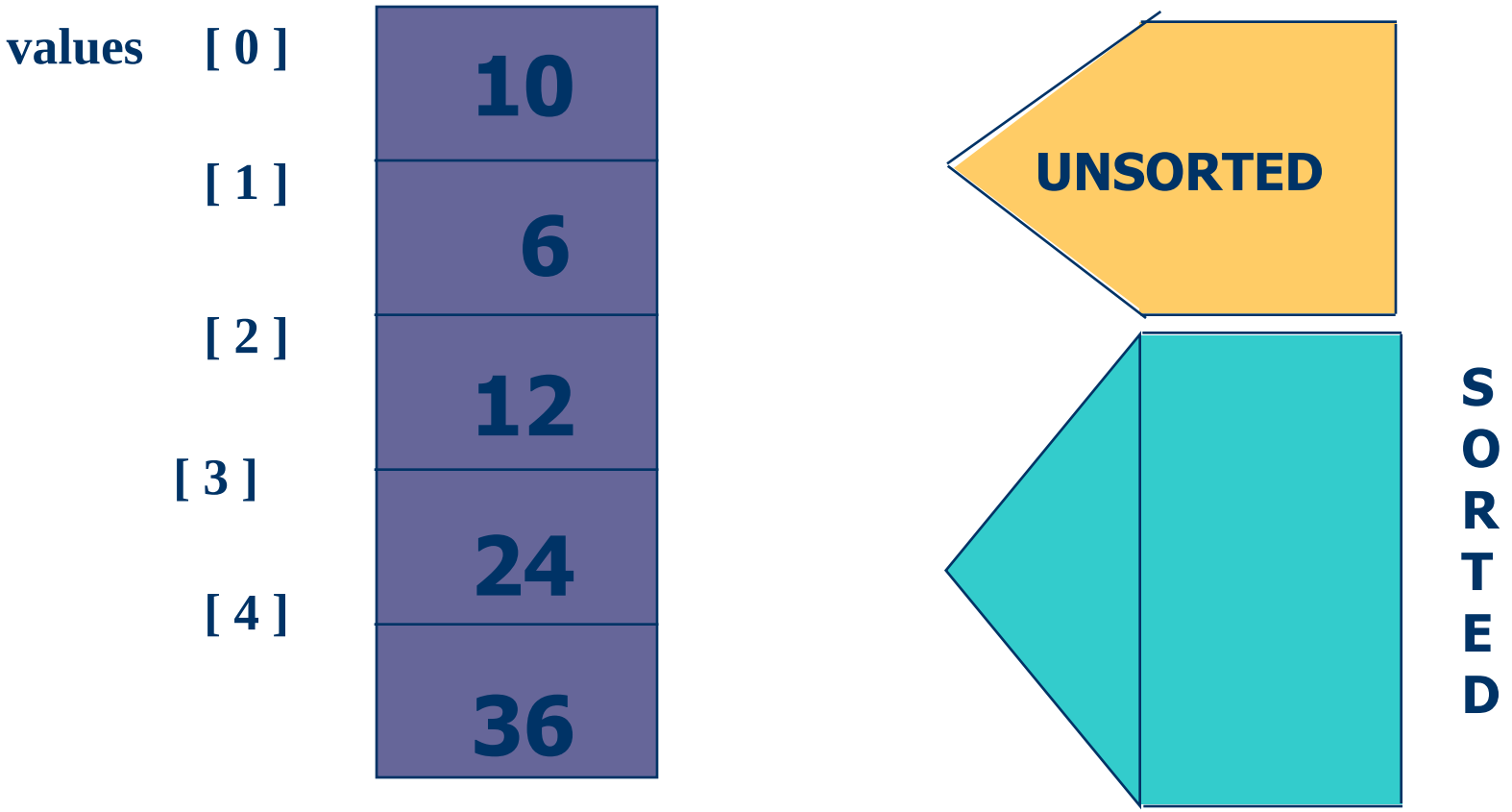
Ταξινόμηση με επιλογή



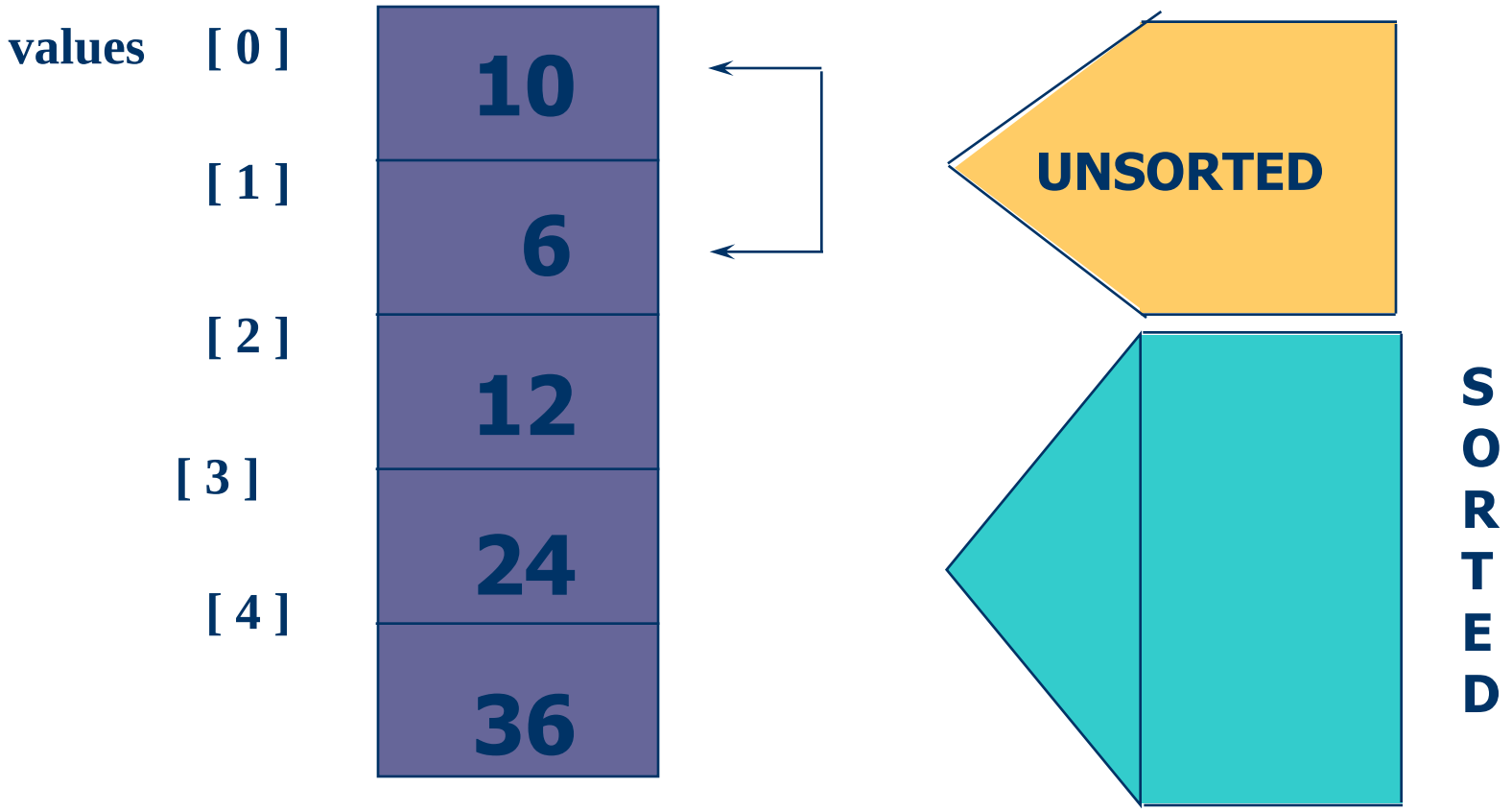
Ταξινόμηση με επιλογή



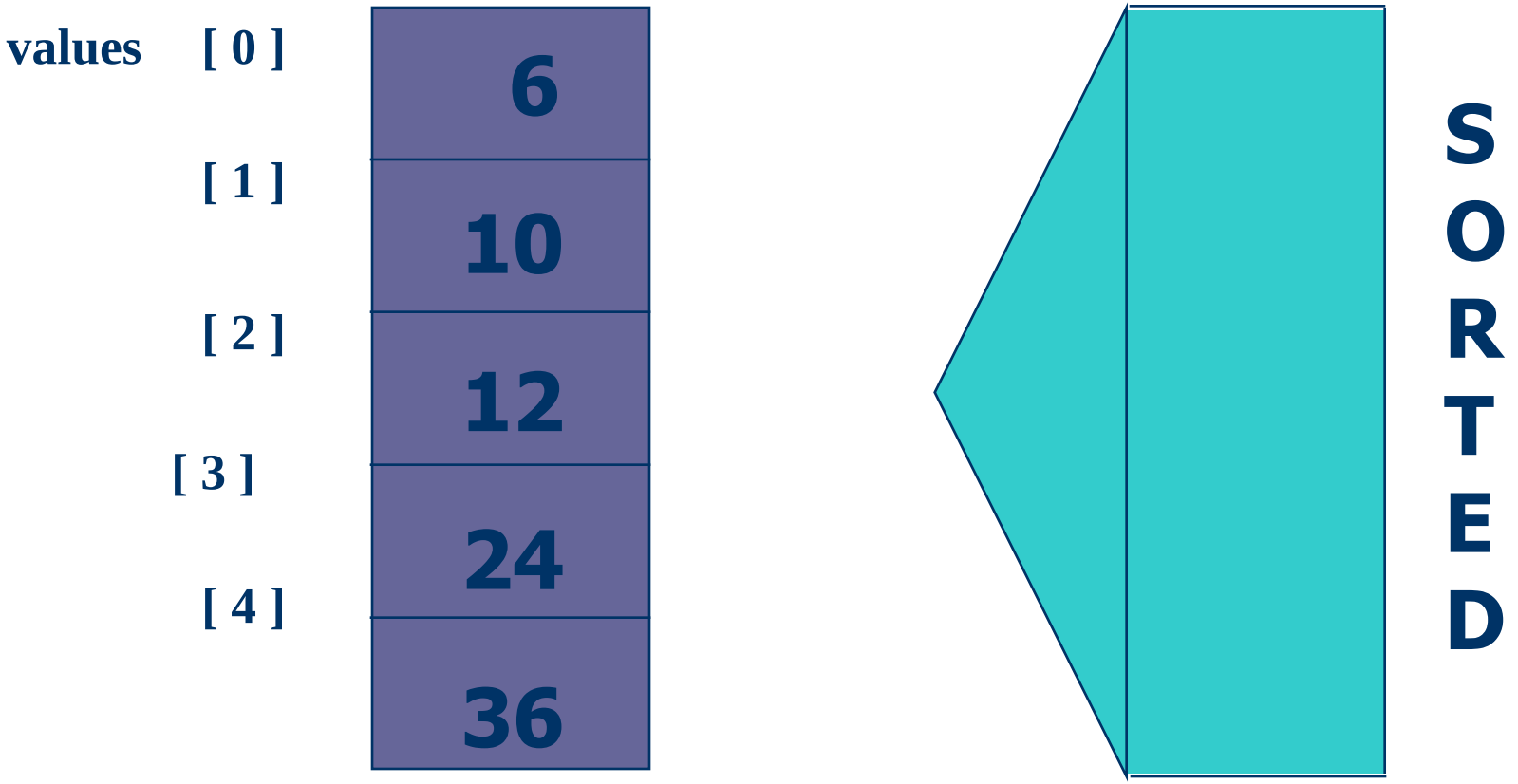
Ταξινόμηση με επιλογή



Ταξινόμηση με επιλογή

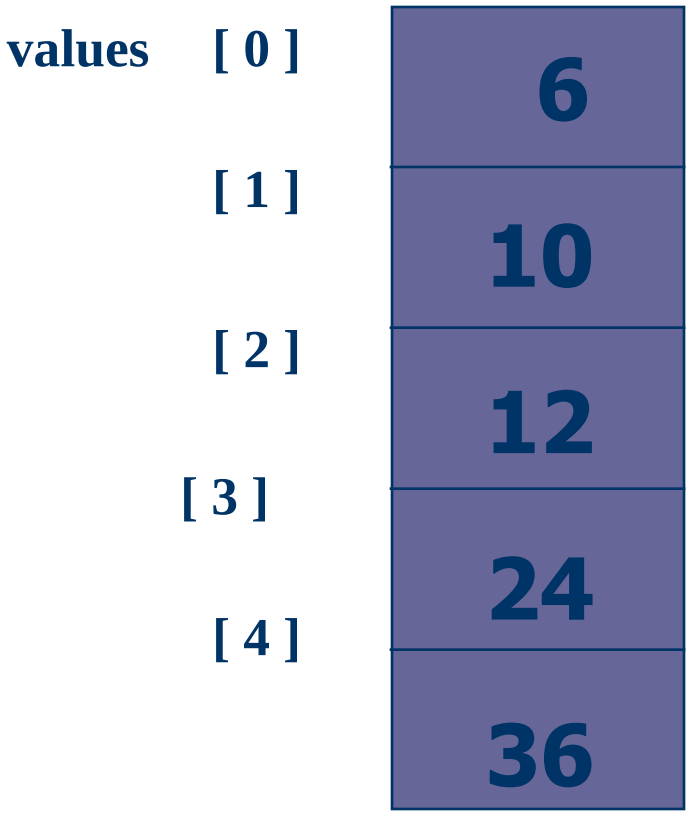


Ταξινόμηση με επιλογή



Ταξινόμηση με επιλογή:

Απαρίθμηση συγκρίσεων (υπολογιστικό κόστος)



4 compares for values[4]

3 compares for values[3]

2 compares for values[2]

1 compare for values[1]

= 4 + 3 + 2 + 1

Min Index

```
minIndex = 0;
minValue = arr[minIndex];
 for (j = 1; j < n; j++)
        if (arr[j] < minValue)
        {
                minIndex = j;
                minValue = arr[j];
        }
```

Swap

- int a = 6;
- int b = 5;
- int tmp = a;
- a = b;
- b = tmp;

Selection sort

```
void selectionSort(int arr[], int n)
{
    int i, j, minIndex, tmp;
    for (i = 0; i < n - 1; i++)
    {
        minIndex = i;
        for (j = i + 1; j < n; j++)
            if (arr[j] < arr[minIndex])
                minIndex = j;
        if (minIndex != i)
        {
            tmp = arr[i];
            arr[i] = arr[minIndex];
            arr[minIndex] = tmp;
        }
    }
}
```

QSort: Ψευδοκώδικας

- QSort(Πίνακας Π, δείκτης Α, δείκτης Τ)
  Ταξινομεί τον υποπίνακα Π[Α] έως Π[Τ]
- Αν Α ≥ Τ, επέστρεψε
- Διαφορετικά,
  - Διάλεξε ένα στοιχείο του πίνακα (pivot), έστω το μεσαίο Λ = Π[(Α+Τ)/2]
  - Βρες την θέση Θ του Λ που θα έχει στην τελική ταξινόμηση
  - Μετέφερε τα στοιχεία του Π[Α] έως και Π[Τ] έτσι ώστε:
    - Π[Χ] ≤ Λ, αν Χ < Θ
    - Π[Χ] > Λ, αν Χ > Θ
- QSort(Π, Α, Θ-1)
- QSort(Π, Θ+1, Τ)

Qsort

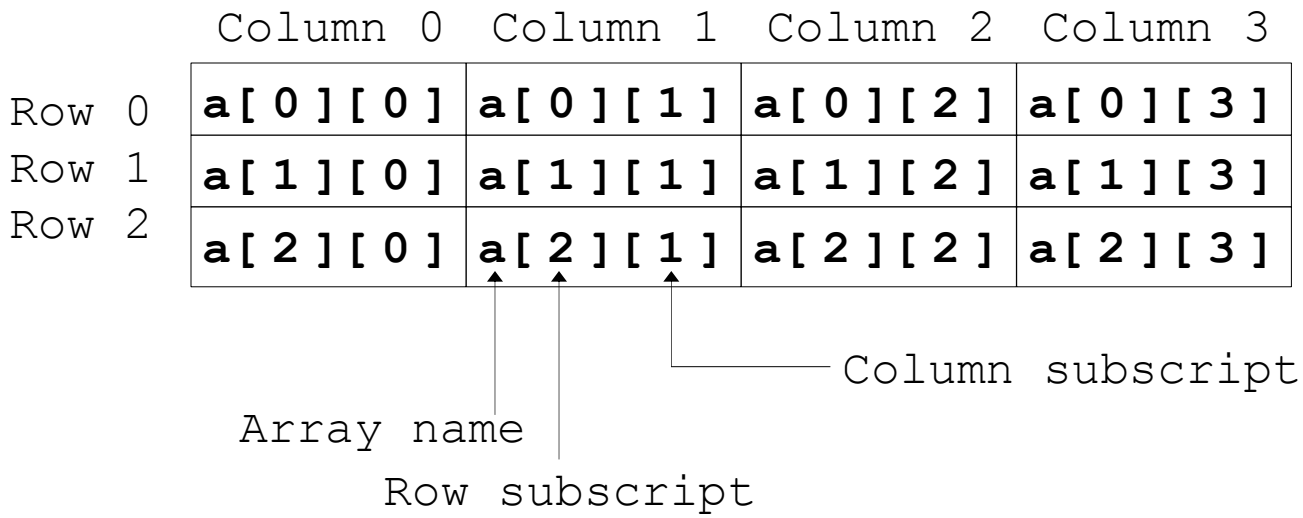
```
void qsort(int v[], int left, int right)
{
   int i, last;
   if(left >= right)
       return;
   swap(v, left, (left + right)/2);
   last = left;
   for (i = left+1; i <= right; i++)
       if (v[i] < v[left])
              swap(v, ++last, i);
   swap(v, left, last);
   qsort(v, left, last-1);
   qsort(v, last+1, right);
}
void swap(int v[], int i, int j)
{
   int temp;
   temp = v[i];
   v[i] = v[j];
   v[j] = temp;
}
```

ΗΥ-150
Προγραμματισμός

Πολυδιάστατοι Πίνακες

Πίνακες πολλών διαστάσεων

- Πίνακες πολλών διαστάσεων
  - Πίνακες με γραμμές και στήλες (**m** X **n** πίνακας)
  - Όπως και στα μαθηματικά: πρώτα γραμμή και μετά στήλη



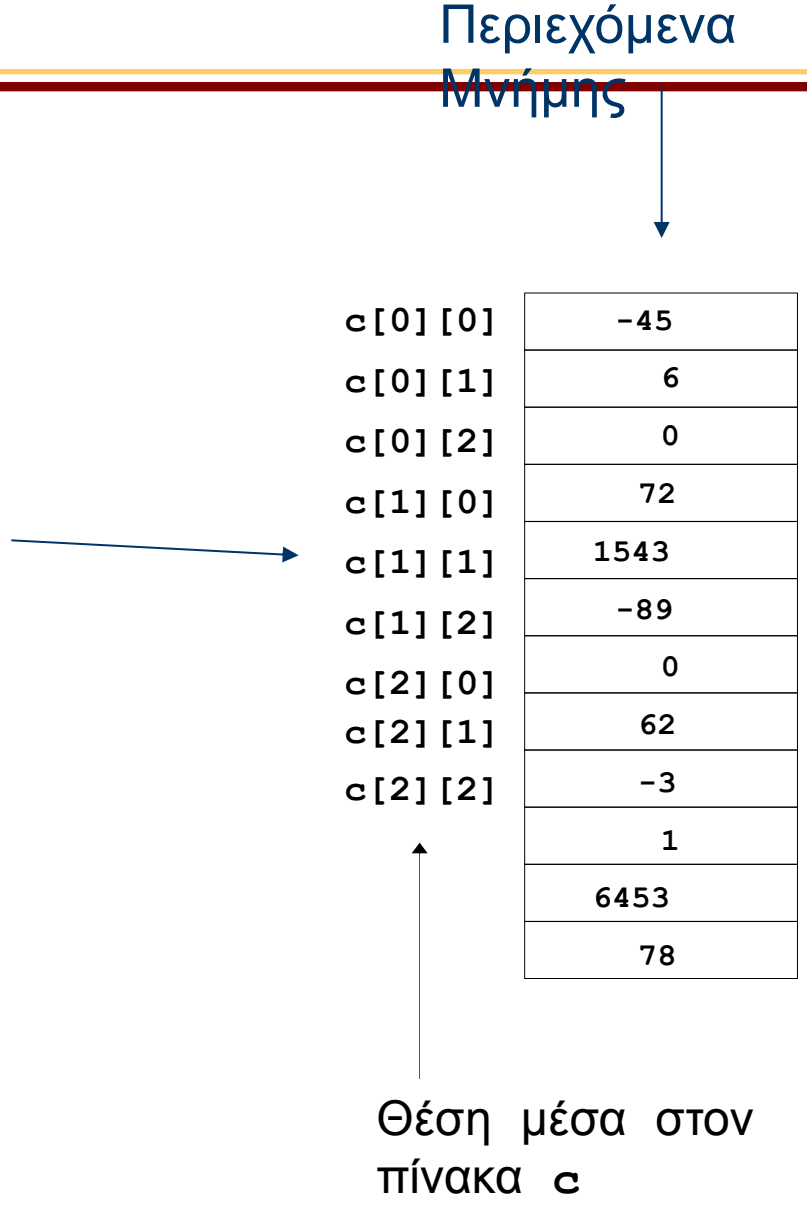
- Ένας διδιάστατος πίνακας ορίζεται ως εξής:

  τύπος όνομα[πλήθος_στοιχείων_1][πλήθος_στοιχείων_2];
- Πίνακες περισσότερων διαστάσεων ορίζονται ανάλογα. Ένας ακέραιος 6 x 8 διδιάστατος πίνακας είναι ο **int** a[6][8];
- Το στοιχείο π.χ. (3,2) του πίνακα αυτού είναι προσπελάσιμο ως a[3][2] .
- Απόδοση αρχικών τιμών γίνεται ανά γραμμή, ως εξής:

int b[2][3] = **{ {**0, 1, 2**}**, **{**3, 4, 5**} }**;

Δισδιάστατοι Πίνακες

- Δήλωση: int c[3][3];
- Χρήση: c[k][j] = m;
- Αποθήκευση στη μνήμη



Πίνακες πολλών διαστάσεων

- Αρχικοποίηση
  - **int b[2][2] = { { 1, 2 }, { 3, 4 } };**
  - {} διαχωρίζουν γραμμές του πίνακα
  - Αν δεν είναι αρκετά τα στοιχεία, τα υπόλοιπα γίνονται 0

    int b[2][2] = { { 1 }, { 3, 4 } };

1	2
3	4

1	0
3	4

Παράδειγμα 2Δ πίνακα

```
#include <stdio.h>

int main()
{
	int magic[3][3] =
	{ {8, 1, 6},
		{3, 5, 7},
		{4, 9, 2}
	};

	int i, j, sum;
	int sumrows[3];
	int sumcolumns[3];
	int sumdiagonal1;
	int sumdiagonal2;

	// Print the magic square
	for (i=0; i < 3; i++)
	{
		for (j = 0; j < 3; j++)
		{
			cout << magic[i][j];
		}
		cout << endl;
	}
```

```
// Now check that it is a magic square:
	// all rows and columns need to add to 15
	sumdiagonal1 = 0;
	sumdiagonal2 = 0;
	for (i = 0 ; i < 3; i++)
	{
		sumrows[i] = 0;
		sumcolumns[i] = 0;
		for (j = 0 ; j < 3; j++)
		{
			sumrows[i] += magic[i][j];
			sumcolumns[i]+= magic[j][i];
		}

		sumdiagonal1 += magic[i][i];
		sumdiagonal2 += magic[i][2-i];
	}

	for (i=0; i < 3 ; i++)
	{
		cout<<"Sum Row "<<i<<" is " <<sumrows[i]<<endl;
		cout<<"Sum Column "<<i<<" is " sumcolumns[i]<<endl;
	}
	cout<<"Sum diagonal 1 is "<<sumdiagonal1 <<endl;
	cout<<"Sum diagonal 2 is "<<sumdiagonal2<<endl;

	return 0;
}
```

Πολυδιάστατους Πίνακες ως Ορίσματα

- Πάντα πρέπει να δηλώνεται το μέγεθος κάθε διάστασης, εκτός από την πρώτη διάσταση (αριθμός γραμμών)

```
int f(int a[][10])
{
        …
}
int main(void)
{
   int b[40][10];
   f(b);
}
```

- Γιατί μας υποχρεώνει η γλώσσα σε αυτόν τον περιορισμό; Σκεφτείτε.

Υπολογισμός Mean, Median and Mode

- Mean – average
  - 1, 1, 2, 2, 4
  - Mean = (1+1+2+2+4) / 5 = 2.0
- Median – number in middle of sorted list
  - 1, 2, 3, 4, 5
  - 3 is the median
- Mode – number that occurs most often
  - 1, 1, 1, 2, 3, 3, 4, 5
  - 1 is the mode

```
1  /*
2     Double-subscripted array example */
3  #include <stdio.h>
4  #define STUDENTS 3
5  #define EXAMS 4
6
7  int minimum( const int [][ EXAMS ], int, int );
8  int maximum( const int [][ EXAMS ], int, int );
9  double average( const int [], int );
10 void printArray( const int [][ EXAMS ], int, int );
11
12 int main()
13 {
14    int student;
15    const int studentGrades[ STUDENTS ][ EXAMS ] =
16       { { 77, 68, 86, 73 },
17         { 96, 87, 89, 78 },
18         { 70, 90, 86, 81 } };
19
20    cout << "The array is:" << endl;
21    printArray( studentGrades, STUDENTS, EXAMS );
22    cout<<"\n\nLowest grade: " << minimum( studentGrades, STUDENTS, EXAMS ) << endl;
23    cout<<"Highest grade: " << maximum( studentGrades, STUDENTS, EXAMS ) << "\n",
24 );
25
26    for ( student = 0; student <= STUDENTS - 1; student++ )
27       cout << "The average grade for student " << student
28                << " is " << average( studentGrades[ student ], EXAMS ) << endl;
29
30
31    return 0;
32 }
```

Each row is a particular student, each column is the grades on the exam.

Notice! Each studentGrades[i] is a one-dimensional array with the grades of student i.

```
33
34 /* Find the minimum grade */
35 int minimum( const int grades[][ EXAMS ],
36              int pupils, int tests )
37 {
38    int i, j, lowGrade = 100;
39
40    for ( i = 0; i <= pupils - 1; i++ )
41       for ( j = 0; j <= tests - 1; j++ )
42          if ( grades[ i ][ j ] < lowGrade )
43             lowGrade = grades[ i ][ j ];
44
45    return lowGrade;
46 }
47
48 /* Find the maximum grade */
49 int maximum( const int grades[][ EXAMS ],
50              int pupils, int tests )
51 {
52    int i, j, highGrade = 0;
53
54    for ( i = 0; i <= pupils - 1; i++ )
55       for ( j = 0; j <= tests - 1; j++ )
56          if ( grades[ i ][ j ] > highGrade )
57             highGrade = grades[ i ][ j ];
58
59    return highGrade;
60 }
61
62 /* Determine the average grade for a particular exam */
63 double average( const int setOfGrades[], int tests )
64 {
```

```
65    int i, total = 0;
66
67    for ( i = 0; i <= tests - 1; i++ )
68       total += setOfGrades[ i ];
69
70    return ( double ) total / tests;
71 }
72
73 /* Print the array */
74 void printArray( const int grades[][ EXAMS ],
75                  int pupils, int tests )
76 {
77    int i, j;
78
79    cout<<"                 [0]  [1]  [2]  [3]";
80
81    for ( i = 0; i <= pupils - 1; i++ ) {
82       cout << "\nstudentGrades[" << i << "] ";
83
84       for ( j = 0; j <= tests - 1; j++ )
85          cout << grades[ i ][ j ];
86    }
87 }
```

Program Output

```
The array is:
                 [0]   [1]   [2]   [3]
studentGrades[0] 77    68    86    73
studentGrades[1] 96    87    89    78
studentGrades[2] 70    90    86    81

Lowest grade: 68
Highest grade: 96
The average grade for student 0 is 76.00
The average grade for student 1 is 87.50
The average grade for student 2 is 81.75
```

Υπολογισμός Mean, Median and Mode

- Mean – average
  - 1, 1, 2, 2, 4
  - Mean = (1+1+2+2+4) / 5 = 2.0
- Median – number in middle of sorted list
  - 1, 2, 3, 4, 5
  - 3 is the median
- Mode – number that occurs most often
  - 1, 1, 1, 2, 3, 3, 4, 5
  - 1 is the mode

```c
/* Fig. 6.16: fig06_16.c
   This program introduces the topic of survey data analysis.
   It computes the mean, median, and  mode of the data */
#include <stdio.h>
#define SIZE 99

void mean( const int [] );
void median( int [] );
void mode( int [], const int [] ) ;
void bubbleSort( int [] );
void printArray( const int [] );

int main()
{
   int frequency[ 10 ] = { 0 };
   int response[ SIZE ] =
      { 6, 7, 8, 9, 8, 7, 8, 9, 8, 9,
        7, 8, 9, 5, 9, 8, 7, 8, 7, 8,
        6, 7, 8, 9, 3, 9, 8, 7, 8, 7,
        7, 8, 9, 8, 9, 8, 9, 7, 8, 9,
        6, 7, 8, 7, 8, 7, 9, 8, 9, 2,
        7, 8, 9, 8, 9, 8, 9, 7, 5, 3,
        5, 6, 7, 2, 5, 3, 9, 4, 6, 4,
        7, 8, 9, 6, 8, 7, 8, 9, 7, 8,
        7, 4, 4, 2, 5, 3, 8, 7, 5, 6,
        4, 5, 6, 1, 6, 5, 7, 8, 7 };

   mean( response );
   median( response );
   mode( frequency, response );
   return 0;
}
```

```
33
34 void mean( const int answer[] )
35 {
36    int j, total = 0;
37
38    cout << "********" << "  Mean" << "********" << endl;
39
40    for ( j = 0; j <= SIZE - 1; j++ )
41       total += answer[  j  ];
42
43    cout << "The mean is the average value of the data\n" <<
44            "items. The mean is equal to the total of\n" <<
45            "all the data items divided by the number\n" <<
46            "of data items ( " << SIZE <<"). The mean value for\n"<<
47            "this run is: " << total << "/" << SIZE << " = " <<
48            (  double  ) total / SIZE << endl);
49 }
50
51 void median( int answer[] )
52 {
53    cout << "********" << " Median" << "********" << endl;
54
55    cout << "The unsorted array of responses is" << endl;
56
57    printArray( answer );
58    bubbleSort( answer );
59    cout << "\n\nThe sorted array is" ;
60    printArray( answer );
61    cout( "\n\nThe median is element " << SIZE/2 << " of\n"
62            "the sorted " << SIZE " << " element array.\n"
63            "For this run the median is "answer[SIZE/2] << "\n\n";
64
```

```
65 }
66
67 void mode( int freq[], const int answer[] )
68 {
69    int rating, j, h, largest = 0, modeValue = 0;
70
71    cout << "\n" << "********\n" <<  "Mode\n" << "********\n";

73
74    for ( rating = 1; rating <= 9; rating++ )
75       freq[ rating ] = 0;
76
77    for ( j = 0; j <= SIZE - 1; j++ )
78       ++freq[ answer[ j ] ];
79
80
81    cout << "Response    Frequency    Histogram" << endl;
82`
83
84    for ( rating = 1; rating <= 9; rating++ ) {
85       cout << rating << freq[ rating ];
86
87       if ( freq[ rating ] > largest ) {
88          largest = freq[ rating ];
89          modeValue = rating;
90       }
91
92       for ( h = 1; h <= freq[ rating ]; h++ )
93          cout << "*";
94
```

Notice how the subscript in **`frequency[]`** is the value of an element in **`response[]`** (**`answer[]`**)

Print stars depending on value of **`frequency[]`**

```
95        cout << "\n";
96     }
97
98     cout << "The mode is the most frequent value.\n" <<
99       "For this run the mode is " << modeValue << "which occurred" <<
100      largest << " times.\n";
101}
102
103 void bubbleSort( int a[] )
104 {
105    int pass, j, hold;
106
107    for ( pass = 1; pass <= SIZE - 1; pass++ )
108
109       for ( j = 0; j <= SIZE - 2; j++ )
110
111          if ( a[ j ] > a[ j + 1 ] ) {
112             hold = a[ j ];
113             a[ j ] = a[ j + 1 ];
114             a[ j + 1 ] = hold;
115          }
116 }
117
118 void printArray( const int a[] )
119 {
120    int j;
121
122    for ( j = 0; j <= SIZE - 1; j++ ) {
123
124       if ( j % 20 == 0 )
125          cout << "\n" ;
```

Bubble sort: if elements out of order, swap them.

```
126
127       cout << a[ j ] ;
128    }
129 }
```

```
********
 Mean
********
The mean is the average value of the data
items. The mean is equal to the total of
all the data items divided by the number
of data items (99). The mean value for
this run is: 681 / 99 = 6.8788

********
 Median
********
The unsorted array of responses is
7 8 9 8 7 8 9 8 9 7 8 9 5 9 8 7 8 7 8
 6 7 8 9 3 9 8 7 8 7 7 8 9 8 9 8 9 7 8 9
 6 7 8 7 8 7 9 8 9 2 7 8 9 8 9 8 9 7 5 3
 5 6 7 2 5 3 9 4 6 4 7 8 9 6 8 7 8 9 7 8
 7 4 4 2 5 3 8 7 5 6 4 5 6 1 6 5 7 8 7

The sorted array is
 1 2 2 2 3 3 3 3 4 4 4 4 4 5 5 5 5 5 5 5
 5 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 7
 7 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8
 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9

The median is element 49 of
the sorted 99 element array.
For this run the median is 7
```

Program Output

Program Output

```
********
  Mode
********
Response  Frequency        Histogram
       1          1          *
       2          3          ***
       3          4          ****
       4          5          *****
       5          8          ********
       6          9          *********
       7         23          ***********************
       8         27          ***************************
       9         19          *******************
The mode is the most frequent value.
For this run the mode is 8 which occurred 27 times.
```