Πίνακες

Δημήτρης Μητρόπουλος

Τμήμα Διοικητικής Επιστήμης και Τεχνολογίας,
Οικονομικό Πανεπιστήμιο Αθηνών
dimitro@aueb.gr

Υλικό από παρουσιάσεις του Πάνου Λουρίδα και το βιβλίο του K. N. King:

C Programming: A Modern Approach

Επισκόπηση

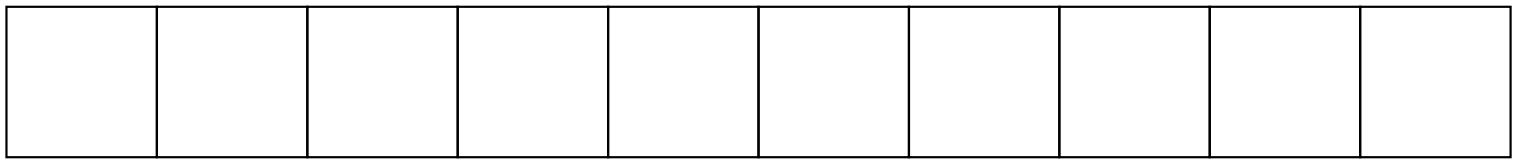
Πίνακες (Arrays)

Βαθμωτές Μεταβλητές έναντι Συνολικών Μεταβλητών

- Μέχρι τώρα, οι μόνες μεταβλητές που έχουμε δει είναι οι *βαθμωτές* (scalar): ικανές να κρατήσουν ένα μόνο στοιχείο δεδομένων.
- Η C υποστηρίζει και τις *συνολικές* (aggregate) μεταβλητές, οι οποίες μπορούν να αποθηκεύσουν συλλογές τιμών.
- Υπάρχουν δύο είδη συνόλων στη C: οι πίνακες και οι δομές.
- Σε αυτό το κεφάλαιο, θα αναλύσουμε τους μονοδιάστατους πίνακες, που παίζουν πολύ μεγαλύτερο ρόλο στη C από τους πολυδιάστατους πίνακες, που επίσης υποστηρίζονται από τη γλώσσα.

Μονοδιάστατοι Πίνακες

- Ένας πίνακας είναι μία δομή δεδομένων που περιλαμβάνει έναν αριθμό τιμών δεδομένων, όλα του ιδίου τύπου.
- Αυτές οι τιμές, γνωστές ως *στοιχεία* (elements), μπορούν να επιλεχθούν μεμονωμένα από τη θέση τους μέσα στον πίνακα.
- Το απλούστερο είδος πίνακα έχει μόνο μία διάσταση.
- Τα στοιχεία ενός μονοδιάστατου πίνακα a είναι εννοιολογικά τοποθετημένα το ένα μετά το άλλο σε μία γραμμή (ή στήλη).



Μονοδιάστατοι Πίνακες

- Για να δηλώσουμε έναν πίνακα, πρέπει να καθορίσουμε τον *τύπο* των στοιχείων του πίνακα και τον *αριθμό* των στοιχείων:

```
int a[10];
```

- Τα στοιχεία μπορούν να είναι οποιουδήποτε τύπου. Το μήκος του πίνακα μπορεί να είναι κάθε (ακέραια) συνεχής παράσταση.
- Η χρήση μάκρο για να ορίσουμε το μέγεθος ενός πίνακα είναι μια πολύ καλή πρακτική:

```
#define N 10

/* ... */

int a[N];
```

Πίνακες (Arrays)

Δεικτοδότηση Πινάκων (Array Subscripting)

- Για να έχουμε πρόσβαση σε ένα στοιχείο πίνακα, γράφουμε το όνομα του πίνακα ακολουθούμενο από μία ακέραια τιμή σε αγκύλες.
- Αυτό ονομάζεται *δεικτοδότηση* (subscripting, indexing) του πίνακα.
- Τα στοιχεία ενός πίνακα με μέγεθος n κατατάσσονται σε πίνακα από 0 έως `n-1`.
- Εάν το a είναι ένας πίνακας μεγέθους 10, τα στοιχεία του ορίζονται από `a[0], a[1], ..., a[9]`:

a[0] a[1] a[2] a[3] a[4] a[5] a[7] a[8] a[9]

Δεικτοδότηση Πινάκων (Array Subscripting)

- Παραστάσεις του τύπου `a[i]` είναι lvalues και άρα μπορούν να χρησιμοποιηθούν με τον ίδιο τρόπο όπως οι κανονικές μεταβλητές:

```
a[0] = 1;
printf("%d\n", a[5]);
++a[i];
```

- Γενικώς, εάν ένας πίνακας περιέχει στοιχεία τύπου T, τότε κάθε στοιχείο του πίνακα αντιμετωπίζεται σαν να ήταν μεταβλητή τύπου T.

Δεικτοδότηση Πινάκων (Array Subscripting)

- Πολλά προγράμματα περιέχουν βρόχους **for** η δουλειά των οποίων είναι να εκτελέσουν κάποια λειτουργία σε κάθε στοιχείο ενός πίνακα.
- Παραδείγματα τυπικών λειτουργιών σε πίνακα a μεγέθους n:

```
for (i = 0; i < n; i++)
    a[i] = 0;                   /* clears a */

for (i = 0; i < n; i++)
    scanf("%d", &a[i]);         /* reads data into a */

for (i = 0; i < n; i++)
    sum += a[i];                /* sums the elements of a */
```

Δεικτοδότηση Πινάκων (Array Subscripting)

- Η C δεν απαιτεί να ελέγχονται τα όρια των δεικτών. Εάν ένας δείκτης βγει εκτός των ορίων του πίνακα, η συμπεριφορά του προγράμματος είναι απροσδιόριστη.
- Ένα συχνό λάθος: να ξεχάσουμε ότι ένας πίνακας με n στοιχεία δεικτοδοτείται από 0 σε n-1, και όχι από 1 έως n:

```
int a[10], i;

for (i = 1; i <= 10; i++)
    a[i] = 0;
```

Σε μερικούς μεταγλωττιστές, αυτή η παράσταση **for** θα προκαλέσει μια ατέλειωτη επανάληψη.

Δεικτοδότηση Πινάκων (Array Subscripting)

- Ένας δείκτης πίνακα μπορεί να είναι μία ακέραια παράσταση:

```
a[i+j*10] = 0;
```

- Η παράσταση μπορεί να έχει παράπλευρες συνέπειες (side effects):

```
i = 0;
while (i < n)
    a[i++] = 0;
```

Δεικτοδότηση Πινάκων (Array Subscripting)

- Χρειάζεται προσοχή όταν ένας δείκτης πίνακα έχει παράπλευρες συνέπειες:

```
i = 0;
while (i < n)
    a[i] = b[i++];
```

- Η παράσταση `a[i] = b[i++]` διαβάζει την τιμή του `i` και επίσης μεταβάλλει το `i`, προκαλώντας απροσδιόριστη συμπεριφορά.
- Το πρόβλημα μπορεί να αποφευχθεί απομονώνοντας την αύξηση από το δείκτη.

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

Πρόγραμμα: Αντιστροφή Σειράς Αριθμών

- Το πρόγραμμα `reverse.c` ζητάει από το χρήστη να εισάγει μια σειρά αριθμών, και στη συνέχεια γράφει τους αριθμούς στην αντίστροφη σειρά:

```
Enter 10 numbers: 34 82 49 102 7 94 23 11 50 31
In reverse order: 31 50 11 23 94 7 102 49 82 34
```

- Το πρόγραμμα αποθηκεύει τους αριθμούς σε έναν πίνακα όσο αυτοί διαβάζονται, στη συνέχεια διατρέχει τον πίνακα ανάποδα, και τυπώνει τα στοιχεία ένα προς ένα.

reverse.c

```c
/* Reverses a series of numbers */
#include <stdio.h>

#define N 10

int main(void)
{
    int a[N], i;

    printf("Enter %d numbers: ", N);
    for (i = 0; i < N; i++)
        scanf("%d", &a[i]);

    printf("In reverse order:");
    for (i = N - 1; i >= 0; i--)
        printf(" %d", a[i]);
    printf("\n");

    return 0;
}
```

Πίνακες (Arrays)

Αρχικοποίηση Πίνακα

- Σε έναν πίνακα, όπως σε κάθε άλλη μεταβλητή, μπορεί να δοθεί μία αρχική τιμή τη στιγμή της δήλωσής του.
- Η πιο κοινή μορφή αρχικοποιητή πίνακα είναι η λίστα σταθερών παραστάσεων εσώκλειστων σε αγκύλες και χωρισμένων με κόμματα.

```
int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

Αρχικοποίηση Πίνακα

- Εάν ο αρχικοποιητής είναι μικρότερος από τον πίνακα, τα υπολειπόμενα στοιχεία του πίνακα παίρνουν την τιμή 0:

```
int a[10] = {1, 2, 3, 4, 5, 6};
/* initial value of a is {1, 2, 3, 4, 5, 6, 0, 0, 0, 0} */
```

- Χρησιμοποιώντας αυτήν την ιδιότητα, μπορούμε εύκολα να αρχικοποιήσουμε έναν πίνακα με όλα μηδενικά:

```
int a[10] = {0};
/* initial value of a is {0, 0, 0, 0, 0, 0, 0, 0, 0, 0} */
```

  Υπάρχει ένα τουλάχιστον 0 μέσα στις αγκύλες επειδή δεν επιτρέπεται ο αρχικοποιητής να είναι τελείως άδειος.

- Επίσης, δεν επιτρέπεται ο αριθμός των στοιχείων να είναι μεγαλύτερος από τον πίνακα που αρχικοποιείται.

Αρχικοποίηση Πίνακα

- Εάν υπάρχει αρχικοποιητής, τότε το μέγεθος του πίνακα μπορεί να παραλείπεται:

```
int a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

- Ο μεταγλωττιστής χρησιμοποιεί το μέγεθος του αρχικοποιητή για να καθορίσει πόσο μεγάλος θα είναι ο πίνακας.

Προσδιοριστές Αρχικοποίησης (C99)

- Συχνά σχετικά λίγα στοιχεία ενός πίνακα χρειάζονται ρητώς αρχικοποίηση. Τα υπόλοιπα στοιχεία μπορούν να πάρουν προεπιλεγμένες τιμές.
- Για παράδειγμα:

```
int a[15] =
    {0, 0, 29, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 48};
```

- Για ένα μεγάλο πίνακα, η αρχικοποίηση με αυτόν τον τρόπο είναι κουραστική και επιρρεπής σε λάθη.

Προσδιοριστές Αρχικοποίησης (C99)

- Στη C99 οι *προσδιοριστές αρχικοποίησης* (designated initializers) μπορούν να χρησιμοποιηθούν για να λύσουν αυτό το πρόβλημα.
- Να πώς θα μπορούσαμε να ξαναγράψουμε το προηγούμενο παράδειγμα χρησιμοποιώντας προσδιοριστές αρχικοποίησης:

```
int a[15] = {[2] = 29, [9] = 7, [14] = 48};
```

- Κάθε αριθμός μέσα σε αγκύλες ονομάζεται *προσδιοριστής* (designator).

Προσδιοριστές Αρχικοποίησης (C99)

- Οι προσδιοριστές αρχικοποίησης είναι συντομότεροι και πιο εύκολο να διαβαστούν (τουλάχιστον σε κάποιους πίνακες).
- Επίσης, η σειρά με την οποία παρατίθενται τα στοιχεία δεν έχει πλέον σημασία.
- Ένας άλλος τρόπος να γράψουμε το προηγούμενο παράδειγμα:

```
int a[15] = {[14] = 48, [9] = 7, [2] = 29};
```

Προσδιοριστές Αρχικοποίησης (C99)

- Οι προσδιοριστές πρέπει να είναι ακέραιες συνεχείς παραστάσεις.
- Εάν ο πίνακας που αρχικοποιείται έχει μέγεθος n, κάθε προσδιοριστής πρέπει να είναι μεταξύ 0 και `n-1`.
- Εάν το μέγεθος ενός πίνακα παραλειφθεί, ένας προσιοριστής μπορεί να είναι οποιοσδήποτε μη αρνητικός ακέραιος.
  - Ο μεταγλωττιστής θα συμπεράνει το μέγεθος του πίνακα από το μεγαλύτερο προσδιοριστή.
- Ο ακόλουθος πίνακας θα έχει 24 στοιχεία:

```
int b[] = {[5] = 10, [23] = 13, [11] = 36, [15] = 29};
```

Προσδιοριστές Αρχικοποίησης (C99)

- Μία αρχικοποίηση μπορεί να χρησιμοποιεί και την παλαιότερη (εάν-ένα στοιχείο) και την νεότερη (προσδιορισμένη) τεχνική:

```c
int c[10] = {5, 1, 9, [4] = 3, 7, 2, [8] = 6};
```

Πρόγραμμα: Έλεγχος Αριθμού για Επαναλαμβανόμενα Ψηφία

- Το πρόγραμμα `repdigit.c` ελέγχει κατά πόσον κάποιο από τα ψηφία ενός αριθμού εμφανίζεται περισσότερες από μία φορά. Μετά την εισαγωγή του αριθμού από το χρήστη, το πρόγραμμα τυπώνει είτε `Repeated digit` είτε `No repeated digit`:

  ```
  Enter a number: 28212
  Repeated digit
  ```

- Ο αριθμός 28212 έχει έναν επαναλαμβανόμενο αριθμό (2). Ένας αριθμός όπως ο 9357 δεν έχει.

Πρόγραμμα: Έλεγχος Αριθμού για Επαναλαμβανόμενα Ψηφία

- Το πρόγραμμα χρησιμοποιεί έναν πίνακα 10 θέσεων για να παρακολουθεί ποια ψηφία εμφανίζονται στον αριθμό.
- Αρχικά, κάθε στοιχείο του πίνακα `digit_seen` είναι 0 (ψευδές).
- Όταν δοθεί ένας αριθμός n, το πρόγραμμα εξετάζει τα ψηφία του ένα προς ένα, αποθηκεύοντας το τρέχον ψηφίο σε μία μεταβλητή που ονομάζεται `digit`.
  - Εάν το `digit_seen[digit]` είναι αληθές, το `digit` εμφανίζεται τουλάχιστον δύο φορές στο n.
  - Εάν το `digit_seen[digit]` είναι ψευδές, το `digit` δεν έχει ξαναεμφανιστεί, επομένως το πρόγραμμα θέτει το `digit_seen[digit]` σε 1 (αληθές) και συνεχίζει.

repdigit.c

```c
/* Checks numbers for repeated digits */
#include <stdio.h>

int main(void)
{
    int digit_seen[10] = {0};
    int digit;
    long n;

    printf("Enter a number: ");
    scanf("%ld", &n);
    while (n > 0) {
        digit = n % 10;
        if (digit_seen[digit])
            break;
        digit_seen[digit] = 1;
        n /= 10;
    }
```

repdigit.c

```
    if (n > 0)
        printf("Repeated digit\n");
    else
        printf("No repeated digit\n");

    return 0;
}
```

Πίνακες (Arrays)

Χρήση του τελεστή `sizeof` με Πίνακες

- Ο τελεστής **`sizeof`** μπορεί να μας δώσει το μέγεθος ενός πίνακα (σε bytes).
- Εάν `a` είναι ένας πίνακας 10 ακεραίων, τότε το **`sizeof`**`(a)` είναι 40 (θεωρώντας ότι κάθε ακέραιος απαιτεί τέσσερα bytes).
- Μπορούμε επίσης να χρησιμοποιήσουμε το **`sizeof`** για να μετρήσουμε το μέγεθος ενός στοιχείου πίνακα, π.χ. `a[0]`.
- Διαιρώντας το μέγεθος του πίνακα με το μέγεθος του στοιχείου παίρνουμε το μήκος του πίνακα:

```
sizeof(a) / sizeof(a[0])
```

Χρήση του τελεστή sizeof με Πίνακες

- Χρησιμοποιώντας αυτήν την παράσταση μπορούμε να βρούμε το μήκος ενός πίνακα που δεν το ξέρουμε.
- Για παράδειγμα, ένας βρόχος που αδειάζει τον πίνακα a:

```
for (i = 0; i < sizeof(a) / sizeof(a[0]); i++)
    a[i] = 0;
```

Σημειώστε ότι ο βρόχος δεν χρειάζεται να τροποποιηθεί εάν το μήκος του πίνακα αλλάξει σε κάποια μελλοντική έκδοση του προγράμματος.

Χρήση του τελεστή `sizeof` με Πίνακες

- Μερικοί μεταγλωττιστές παράγουν ένα μήνυμα προειδοποίησης για την παράσταση `i < sizeof(a) / sizeof(a[0])`.
- Η μεταβλητή `i` έχει πιθανότατα τον τύπο **`int`** (ένας προσημασμένος τύπος), ενώ το **`sizeof`** παράγει μία τιμή του τύπου **`size_t`** (μη προσημασμένος τύπος).
- Συγκρίνοντας έναν προσημασμένο ακέραιο με έναν μη προσημασμένο ακέραιο μπορεί να είναι επικίνδυνο, αλλά σε αυτήν την περίπτωση είναι ασφαλές.

Χρησιμοποιώντας τον τελεστή sizeof με Πίνακες

- Για να αποφύγουμε την προειδοποίηση, μπορούμε να προσθέσουμε ένα cast που μετατρέπει το `sizeof(a) / sizeof(a[0])` σε προσημασμένο ακέραιο:

```
for (i = 0; i < (int) (sizeof(a) / sizeof(a[0])); i++)
    a[i] = 0;
```

- Συχνά είναι χρήσιμο να το κάνουμε αυτό ορίζοντας μια μακροεντολή:

```
#define SIZE ((int) (sizeof(a) / sizeof(a[0])))

for (i = 0; i < SIZE; i++)
    a[i] = 0;
```

Πρόγραμμα: Υπολογισμός Τόκου

- Το πρόγραμμα `interest.c` τυπώνει έναν πίνακα που δείχνει την εξέλιξη \$100 που είναι επενδεδυμένα σε διαφορετικά επιτόκια σε μία σειρά ετών.
- Ο χρήστης θα εισάγει ένα βασικό επιτόκιο, έστω r και τον αριθμό των ετών που θα επενδυθούν τα χρήματα.
- Ο πίνακας θα δείχνει την ετήσια εξέλιξη του κεφαλαίου για τέσσερα διαφορετικά επιτόκια $r+1$, $r+2$, $r+3$, $r+4$, θεωρώντας ότι το κεφάλαιο τοκίζεται μία φορά το χρόνο.

Πρόγραμμα: Υπολογισμός Τόκου

- Παράδειγμα εκτέλεσης:

```
Enter interest rate: 6
Enter number of years: 5

Years      6%      7%      8%      9%     10%
  1     106.00  107.00  108.00  109.00  110.00
  2     112.36  114.49  116.64  118.81  121.00
  3     119.10  122.50  125.97  129.50  133.10
  4     126.25  131.08  136.05  141.16  146.41
  5     133.82  140.26  146.93  153.86  161.05
```

Πρόγραμμα: Υπολογίζοντας τον Τόκο

- Οι αριθμοί στη δεύτερη σειρά εξαρτώνται από τους αριθμούς στην πρώτη σειρά, επομένως έχει νόημα να αποθηκεύεται η πρώτη γραμμή σε έναν πίνακα.
  - Οι τιμές στον πίνακα χρησιμοποιούνται στη συνέχεια για να υπολογιστεί η δεύτερη σειρά.
  - Αυτή η διαδικασία μπορεί να επαναληφθεί για την τρίτη και τις επόμενες σειρές.
- Το πρόγραμμα χρησιμοποιεί εμφωλευμένες εντολές `for`.
  - Ο εξωτερικός βρόχος μετράει από 1 μέχρι τον αριθμό των ετών που έχει ζητήσει ο χρήστης.
  - Ο εσωτερικός βρόχος αυξάνει το επιτόκιο από την χαμηλότερη τιμή στην υψηλότερη τιμή.

interest.c

```c
/* Prints a table of compound interest */

#include <stdio.h>

#define NUM_RATES ((int) (sizeof(value) / sizeof(value[0])))
#define INITIAL_BALANCE 100.00

int main(void)
{
    int i, base_rate, num_years, year;
    double value[5];

    printf("Enter interest rate: ");
    scanf("%d", &base_rate);
    printf("Enter number of years: ");
    scanf("%d", &num_years);
    printf("\nYears");
```

interest.c

```
    for (i = 0; i < NUM_RATES; i++) {
        printf("%6d%%", base_rate + i);
        value[i] = INITIAL_BALANCE;
    }
    printf("\n");

    for (year = 1; year <= num_years; year++) {
        printf("%3d    ", year);
        for (i = 0; i < NUM_RATES; i++) {
            /* new value is old value plus interest */
            value[i] += (base_rate + i) / 100.0 * value[i];
            printf("%7.2f", value[i]);
        }
        printf("\n");
    }

    return 0;
}
```

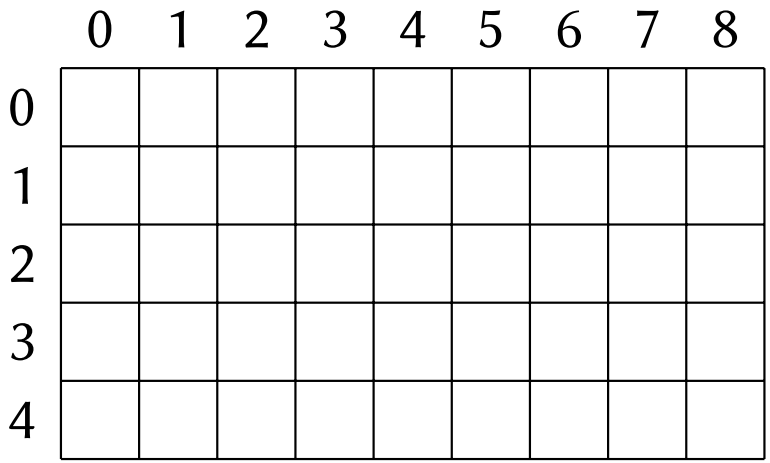
Πίνακες (Arrays)

Πολυδιάστατοι Πίνακες

- Ένας πίνακας μπορεί να έχει οσεσδήποτε διαστάσεις.
- Η ακόλουθη δήλωση, δημιουργεί ένα δισδιάστατο πίνακα (μία μήτρα, σε μαθηματική ορολογία):

```
int m[5][9];
```

- Ο m έχει 5 γραμμές και 9 στήλες. Τόσο οι γραμμές όσο και οι στήλες δεικτοδοτούνται από το 0:

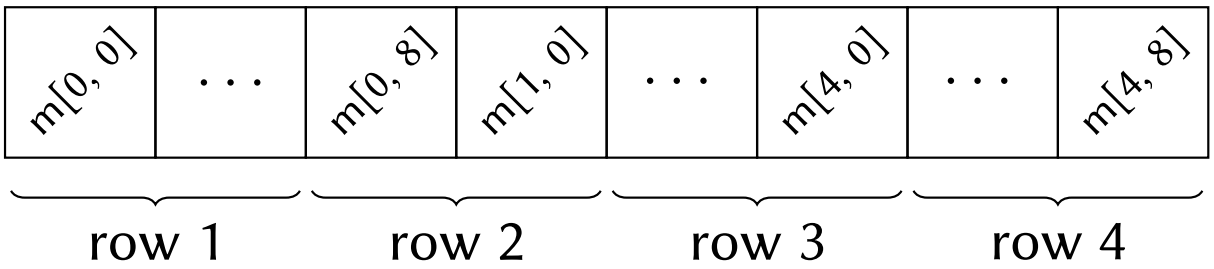


Πολυδιάστατοι Πίνακες

- Για να πάμε σε ένα στοιχείο του `m` στη γραμμή `i` και στη στήλη `j`, πρέπει να γράψουμε `m[i][j]`.
- Η έκφραση `m[i]` προσδιορίζει τη γραμμή `i` του `m` και το `m[i][j]` επιλέγει στη συνέχεια το στοιχείο `j` σε αυτήν τη γραμμή.

Πολυδιάστατοι Πίνακες

- Παρόλο που φανταζόμαστε τους δισδιάστατους πίνακες σαν πραγματικούς πίνακες, στην πράξη δεν αποθηκεύονται στη μνήμη του υπολογιστή με αυτόν τον τρόπο.
- Η C αποθηκεύει τους πίνακες κατά γραμμές, με τη γραμμή 0 πρώτη, μετά τη γραμμή 1, κ.λπ.
- Ο πίνακας `m` αποθηκεύεται ως εξής:



Πολυδιάστατοι Πίνακες

- Ο ιδανικός τρόπος να επεξεργαστούμε πολυδιάστατους πίνακες είναι Εμφωλευμένες εντολές **for**.
- Φανταστείτε το πρόβλημα της αρχικοποίησης ενός πίνακα για χρήση σαν ταυτοτική μήτρα. Ένα ζευγάρι εμφωλευμένων βρόχων **for** είναι άριστη επιλογή:

```
#define N 10

double ident[N][N];
int row, col;

for (row = 0; row < N; row++)
    for (col = 0; col < N; col++)
        if (row == col)
            ident[row][col] = 1.0;
        else
            ident[row][col] = 0.0;
```

Αρχικοποίηση Πολυδιάστατου Πίνακα

- Μπορούμε να αρχικοποίησουμε έναν δισδιάστατο πίνακα εμφωλεύοντας μονοδιάστατες αρχικοποιήσεις:

```
int m[5][9] = {{1, 1, 1, 1, 1, 0, 1, 1, 1},
               {0, 1, 0, 1, 0, 1, 0, 1, 0},
               {0, 1, 0, 1, 1, 0, 0, 1, 0},
               {1, 1, 0, 1, 0, 0, 0, 1, 0},
               {1, 1, 0, 1, 0, 0, 1, 1, 1}};
```

- Με αντίστοιχο τρόπο μπορούμε να αρχικοποιήσουμε πίνακες περισσότερων διαστάσεων.
- Η C προσφέρει μία ποικιλία τρόπων για να να συντομεύσει την αρχικοποίηση πολυδιάστατων πινάκων.

Αρχικοποίηση Πολυδιάστατου Πίνακα

- Εάν ένας αρχικοποιητής δεν είναι αρκετά μεγάλος για να γεμίσει ένας πολυδιάστατος πίνακας, τα υπολειπόμενα στοιχεία παίρνουν την τιμή 0.
- Ο ακόλουθος αρχικοποιητής γεμίζει μόνο τις πρώτες τρεις σειρές του `m`. Οι δύο τελευταίες σειρές θα περιέχουν μηδενικά:

```
int m[5][9] = {{1, 1, 1, 1, 1, 0, 1, 1, 1},
               {0, 1, 0, 1, 0, 1, 0, 1, 0},
               {0, 1, 0, 1, 1, 0, 0, 1, 0}};
```

Αρχικοποίηση Πολυδιάστατου Πίνακα

- Εάν μία εσωτερική λίστα δεν είναι αρκετά μεγάλη για να γεμίσει μία γραμμή, τα υπόλοιπα στοιχεία της γραμμής αρχικοποιούνται με το 0:

```
int m[5][9] = {{1, 1, 1, 1, 1, 0, 1, 1, 1},
               {0, 1, 0, 1, 0, 1, 0, 1},
               {0, 1, 0, 1, 1, 0, 0, 1},
               {1, 1, 0, 1, 0, 0, 0, 1},
               {1, 1, 0, 1, 0, 0, 1, 1, 1}};
```

Αρχικοποίηση Πολυδιάστατου Πίνακα

- Μπορούμε ακόμη και να παραλείψουμε τις εσωτερικές αγκύλες:

```
int m[5][9] = {1, 1, 1, 1, 1, 0, 1, 1, 1,
               0, 1, 0, 1, 0, 1, 0, 1, 0,
               0, 1, 0, 1, 1, 0, 0, 1, 0,
               1, 1, 0, 1, 0, 0, 0, 1, 0,
               1, 1, 0, 1, 0, 0, 1, 1, 1};
```

- Όταν ο μεταγλωττιστής δει αρκετές τιμές για να γεμίσει μία γραμμή, ξεκινάει να γεμίζει την επόμενη.
- Μπορεί να είναι ριψοκίνδυνο να παραλείψουμε τις εσωτερικές αγκύλες, αφού τα επιπλέον στοιχεία (ή ακόμη χειρότερα, τα απόντα στοιχεία) θα επηρεάσουν τον υπολειπόμενο αρχικοποιητή.

Αρχικοποίηση Πολυδιάστατου Πίνακα

- Τα προσδιορισμένα στοιχεία αρχικοποίησης της C99 λειτουργούν και με πολυδιάστατους πίνακες.
- Για να φτιάξουμε τον ταυτοτικό πίνακα 2x2 δίνουμε:

```
double ident[2][2] = {[0][0] = 1.0, [1][1] = 1.0};
```

Όπως συνήθως, όλα τα στοιχεία για τα οποία δεν προσδιορίζονται τιμές θα πάρουν την τιμή 0.

Πίνακες (Arrays)

Σταθεροί Πίνακες

- Ένας πίνακας μπορεί να γίνει «σταθερά» ξεκινώντας τη δήλωσή του με τη λέξη **const**:

```
const char hex_chars[] =
    {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
    'A', 'B', 'C', 'D', 'E', 'F'};
```

- Ένας πίνακας που δηλώνεται ως **const** δεν θα πρέπει να τροποποιείται από το πρόγραμμα.

Σταθεροί Πίνακες

- Πλεονεκτήματα της δήλωση ενός πίνακα ως **const**:
  - Τεκμηριώνει ότι το πρόγραμμα δεν θα αλλάξει τον πίνακα.
  - Βοηθάει τον μεταγλωττιστή να αντιληφθεί λάθη.
- Η **const** δεν περιορίζεται σε πίνακες, αλλά είναι ιδιαιτέρως χρήσιμη για τη δήλωση πινάκων.

Πρόγραμμα: Μοιρασιά Τράπουλας

- Το πρόγραμμα `deal.c` χρησιμοποιεί και δισδιάστατους πίνακες και σταθερούς πίνακες.
- Το πρόγραμμα μοιράζει τυχαία χαρτιά από μία συνηθισμένη τράπουλα.
- Κάθε τραπουλόχαρτο έχει ένα χρώμα (σπαθιά, καρώ, κούπες, ή μπαστούνια) και μία αξία (δύο, τρία, τέσσερα, πέντε, έξι, εφτά, οκτώ, εννέα, δέκα, βαλές, ντάμα, ρίγας ή άσσος).

Πρόγραμμα: Μοιρασιά Τράπουλας

- Ο χρήστης θα καθορίσει πόσα χαρτιά θα του μοιραστούν:

```
Enter number of cards in hand: 5
Your hand: 7c 2s 5d as 2h
```

- Συμβολισμός: c = club (σπαθί) s = spade (μπαστούνι), d = diamond (καρώ), h = heart (κούπα).
- Προβλήματα που πρέπει να λύσουμε:
  - Πώς να επιλέξουμε χαρτιά τυχαία από μία τράπουλα;
  - Πώς να αποφύγουμε να διαλέξουμε το ίδιο χαρτί δεύτερη φορά;

Πρόγραμμα: Μοιρασιά Τράπουλας

- Για να διαλέξουμε χαρτιά τυχαία, θα χρησιμοποιήσουμε διάφορες βιβλιοθήκες συναρτήσεων της C:
  - `time` (από `<time.h>`): επιστρέφει την τρέχουσα ώρα, κωδικοποιημένη σε έναν αριθμό.
  - `srand` (από `<stdlib.h>`): αρχικοποιεί τη γεννήτρια τυχαίων αριθμών της C.
  - `rand` (from `<stdlib.h>`): παράγει έναν ψευδοτυχαίο αριθμό κάθε φορά που καλείται.
- Χρησιμοποιώντας τον τελεστή `%`, μπορούμε να πάρουμε την τιμή που επιστρέφει η `rand`, έτσι ώστε να επιστρέφει τιμές από 0 έως και 3 για το (χρώμα) ή από 0 έως και 12 (αξία).

Πρόγραμμα: Μοιρασιά Τράπουλας

- Ο πίνακας `in_hand` χρησιμοποιείται για να παρακολουθεί ποια χαρτιά έχουν ήδη επιλεχθεί.
- Ο πίνακας έχει 4 γραμμές και 13 στήλες. Κάθε στοιχείο αντιστοιχεί σε ένα από τα 52 χαρτιά στην τράπουλα.
- Όλα τα στοιχεία του πίνακα θα έχουν αρχικά την τιμή 0.
- Κάθε φορά που θα επιλέγεται τυχαία ένα χαρτί, θα ελέγχουμε αν το στοιχείο στον `in_hand` που αντιστοιχεί σε αυτό το χαρτί είναι αληθές (διάφορο του μηδέν) ή ψευδές (ίσο με μηδέν).
  - Εάν είναι αληθές, θα πρέπει να επιλέξουμε άλλο χαρτί.
  - Εάν είναι ψευδές, θα αποθηκεύουμε την τιμή 1 σε αυτό το στοιχείο για να θυμηθούμε αργότερα ότι αυτό το χαρτί έχει ήδη επιλεχθεί.

Πρόγραμμα: Μοιράζοντας ένα Χαρτί από Κάρτες

- Όταν επιβεβαιώσουμε ότι ένα χαρτί είναι «νέο», θα πρέπει να μεταφράσουμε την αξία και το χρώμα σε χαρακτήρες και στη συνέχεια να εμφανίσουμε το χαρτί στην οθόνη.
- Για να μεταφράσουμε το βαθμό και το χρώμα σε μορφή χαρακτήρων, θα ορίσουμε δύο πίνακες χαρακτήρων—έναν για την αξία και έναν για το χρώμα—και στη συνέχεια θα χρησιμοποιήσουμε τους αριθμούς για να δεικτοδοτήσουμε τους πίνακες.
- Αυτοί οι πίνακες δεν θα αλλάζουν κατά τη διάρκεια του προγράμματος, επομένως η δήλωσή τους θα είναι `const`.

deal.c

```c
/* Deals a random hand of cards */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define NUM_SUITS 4
#define NUM_RANKS 13

int main(void)
{
    int in_hand[NUM_SUITS][NUM_RANKS] = { 0 };
    int num_cards, rank, suit;
    const char rank_code[] = {'2','3','4','5','6','7','8',
                              '9','t','j','q','k','a'};
    const char suit_code[] = {'c','d','h','s'};
```

deal.c

```
    srand((unsigned) time(NULL));

    printf("Enter number of cards in hand: ");
    scanf("%d", &num_cards);

    printf("Your hand:");
    while (num_cards > 0) {
        suit = rand() % NUM_SUITS;          /* picks a random suit */
        rank = rand() % NUM_RANKS;          /* picks a random rank */
        if (!in_hand[suit][rank]) {
            in_hand[suit][rank] = 1;
            num_cards--;
            printf(" %c%c", rank_code[rank], suit_code[suit]);
        }
    }
    printf("\n");

    return 0;
}
```

Ασκήσεις

1. Αλλάξτε το πρόγραμμα `repdigit.c` ώστε να δείχνει ποια ψηφία (αν υπάρχουν) εμφανίζονται περισσότερο από μία φορά. Παράδειγμα εκτέλεσης:

```
Enter a number: 939577
Repeated digit(s): 7 9
```

2. Αλλάξτε το πρόγραμμα `repdigit.c` ώστε να δείχνει έναν πίνακα στον οποίο θα φαίνεται πόσες φορές εμφανίζεται κάθε ψηφίο στον αριθμό. Παράδειγμα εκτέλεσης:

```
Enter a number: 41271092
Digit:       0 1 2 3 4 5 6 7 8 9
Occurrences: 1 2 2 0 1 0 0 1 0 1
```

Ασκήσεις

3. Αλλάξτε το πρόγραμμα `interest.c` ώστε να κεφάλαιο να τοκίζεται ανά μήνα, αντί για ανά χρόνο. Η μορφοποίηση της εξόδου του προγράμματος δεν πρέπει να αλλάξει: το πρόγραμμα θα εξακολουθεί να εμφανίζει το κεφάλαιο ανά έτος.

Ασκήσεις

4. Γράψτε ένα πρόγραμμα που ζητάει από το χρήστη να δώσει τα αποτελέσματα 5 τεστ σε 5 φοιτητές. Στη συνέχεια θα υπολογίζει και θα εμφανίζει το συνολικό βαθμό και το μέσο όρο του βαθμού ανά φοιτητή, και το μέσο όρο, τον υψηλότερο βαθμό, και το χαμηλότερο βαθμό ανά τεστ. Παράδειγμα εκτέλεσης:

```
Enter grades for student 1: 8 3 9 0 10
Enter grades for student 2: 3 5 17 1 1
Enter grades for student 3: 2 8 6 23 1
Enter grades for student 4: 15 7 3 2 9
Enter grades for student 5: 6 14 2 6 0
Student: 1 total: 30 avg: 6.00
Student: 2 total: 27 avg: 5.40
Student: 3 total: 40 avg: 8.00
Student: 4 total: 36 avg: 7.20
Student: 5 total: 28 avg: 5.60
Test: 1 avg: 6.80 min: 2 max: 15
Test: 2 avg: 7.40 min: 3 max: 14
Test: 3 avg: 7.40 min: 2 max: 17
Test: 4 avg: 6.40 min: 0 max: 23
Test: 5 avg: 4.20 min: 0 max: 10
```

Ασκήσεις

5. Γράψτε ένα πρόγραμμα που παράγει έναν «τυχαίο περίπατο» (random walk) σε έναν πίνακα 10x10. Ο πίνακας αρχικά θα περιέχει παντού το χαρακτήρα '.'. Το πρόγραμμα θα πηγαίνει από στοιχείο σε στοιχείο, πηγαίνοντας ένα βήμα πάνω, κάτω, αριστερά, ή δεξιά. Τα στοιχεία που θα επισκέπτεται το πρόγραμμα θα γίνονται A, B, ..., Z. Παράδειγμα εκτέλεσης:

```
A . . . . . . . . .
B . . . . . . . . .
C . . . . . . . . .
D E . . . . . . . .
. F . . . . . . . .
. G . . . . . . . .
. H I . M N O . . .
. . J K L . P . . .
. . . . . . Q R S .
. . Z Y X W V U T .
```

Ασκήσεις

Χρησιμοποιήστε τις συναρτήσεις `srand` και `rand` για την παραγωγή τυχαίων αριθμών. Αφού πάρετε έναν τυχαίο αριθμό, ελέγξετε το υπόλοιπο της διαίρεσής του με το 4. Οι δυνατές τιμές, 0, 1, 2, 3, θα δείχνουν την κίνηση (πάνω, κάτω, αριστερά, δεξιά). Πριν αλλάξει θέση, το πρόγραμμα θα ελέγχει ότι η νέα θέση είναι εντός του πίνακα και ότι δεν την έχει επισκεφτεί ήδη. Αν έχουμε επισκεφτεί όλες τις δυνατές θέσεις, τότε το πρόγραμμα σταματάει, ακόμα και αν δεν έχουμε φτάσει μέχρι το Z, όπως συμβαίνει στο παρακάτω παράδειγμα εκτέλεσης:

```
A . . . . . .
B C . . . . . .
G D . . . . . .
F E . . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
```

Τέλος, η συνάρτηση `srand` χρησιμοποιείται για να αρχικοποίησουμε τη γεννήτρια τυχαίων αριθμών. Παιρνόντας κάθε φορά διαφορετική τιμή μέσω της `time(NULL)`, το πρόγραμμα εκτελεί διαφορετικό περίπατο. Δοκιμάστε να δείτε τι συμβαίνει αν στην `srand` δώσουμε μια σταθερή παράμετρο, π.χ.:

```
srand(1);
```