# Tablouri. Particularizare - vectori

## Vectori

Printr-un vector se înţelege o colecţie liniară şi omogenă de date.
Un vector este liniar pentru că datele(elementele) pot fi accesate în mod unic printr-un index.
Un vector este, de asemenea, omogen, pentru că toate elementele sunt de acelaşi tip. În limbajul C, indexul este un număr întreg pozitiv şi indexarea se face începând cu 0.
Declaraţia unei variabile de tip vector se face în felul următor:

```
<tip_elemente> <nume_vector>[<dimensiune>];

int a[100];
float vect[50];

#define MAX 100
...
unsigned long numbers[MAX]


int a[3] = {1, 5, 6};                    // Toate
     cele 3 elemente sunt initializate
float num[] = {1.5, 2.3, 0.2, -1.3};      //
    Compilatorul determina dimensiunea - 4 - a
    vectorului
unsigned short vect[1000] = {0, 2, 4, 6}; // Sunt
    initializate doar primele 4 elemente

char string[100] = {97}; //Sirul va fi initializat
     cu 97 (caracterul 'a') pe prima pozitie si 99
     de 0
int v[100] = {0};        // Vectorul va fi umplut
    cu 0.
```

Elementele neiniţializate pot avea valori oarecare. Elementele se accesează prin expresii de forma:

```
<nume_vector>[<indice>].
```

De exemplu, putem avea:

```
char vect[100];
vect[0] = 1;
vect[5] = 10;

int i = 90;
vect[i] = 15;
vect[i + 1] = 20;
```

La alocarea unui vector, compilatorul nu efectuează nici un fel de iniţializare şi nu furnizează nici un mesaj de eroare dacă un element este folosit înainte de a fi iniţializat. Un program corect va iniţializa, în orice caz, fiecare element înainte de a-l folosi.

## Exemple de programe

Citirea unui vector de intregi de la tastatura:

```
int main()
{
  int a[100], n, i; /* vectorul a are maxim 100 de
       intregi */

  scanf("%d", &n); /* citeste nr de elemente
      vector */

  for (i = 0; i < n; i++) {
    scanf("%d", &a[i]); /* citire elemente vector
        */
  }

  for (i = 0; i < n; i++) {
    printf("%d␣", a[i]); /* scrie elemente vector
        */
  }

  return 0;
}
```

Generarea unui vector cu primele n numere Fibonacci:

```
#include <stdio.h>
int main()
{
  long fib[100] = {1, 1};
  int n, i;

  printf("n␣=␣");
  scanf("%d", &n);

  for (i = 2; i < n; i++) {
    fib[i] = fib[i - 1] + fib[i - 2];
  }
  for (i = 0; i < n; i++) {
    printf("%ld␣", fib[i]);
  }

  return 0;
}
```

Definiţi dimensiunile prin constante şi folosiţi-le pe acestea în locul tastării explicite a valorilor în codul sursă.

```
#define MAX   100

int vect[MAX];
```

va fi de preferat în locul lui:

```
int vect[100];
```

## Cautari

Cautarea secventiala:

```
#define MAX 100
 ...
 int v[MAX], x, i;
 /* initializari */
...
 int found = 0;
for (i = 0; i < MAX; i++) {
  if (x == v[i]) {
    found = 1;
    break;
  }
}
 if (found) {
   printf("Valoarea␣%d␣a␣fost␣gasita␣in␣vector\n",
        x);
} else {
   printf("Valoarea␣%d␣nu␣a␣fost␣gasita␣in␣vector\
       n", x);
}
```

Cautarea binara iterativa:

```
/ cauta elementul x in vectorul sortat v, intre
    pozitiile 0 si n-1  si returneaza pozitia
    gasita sau -1
int binary_search(int n, int v[NMAX], int x) {
  int low = 0, high = n - 1;

  while (low <= high) {
    int middle = low + (high - low) / 2;

    if (v[middle] == x) {
      // Am gasit elementul, returnam pozitia sa
      return middle;
    }

    if (v[middle] < x) {
      // Elementul cautat este mai mare decat cel
          curent, ne mutam in jumatatea
      // cu elemente mai mari
      low = middle + 1;
    } else {
      // Elementul cautat este mai mic decat cel
          curent, ne mutam in jumatatea
      // cu elemente mai mici
      high = middle - 1;
    }
  }

  // Elementul nu a fost gasit
  return -1;
}
```