4.TABLOU

TABLOUL

Structură de date statică.

Definiţie: Fie C o colecţie de elemente de tipul TE şi
[n]={1,2,3,...,n} cu n ∈ N* o mulţime de indici.
Aplicaţia $f : [n_1] \times [n_2] \times \ldots \times [n_k] \rightarrow C$ care ataşează fiecărui tuplu de *k* indici $(i_1, i_2, \ldots, i_k)$ unde $i_j \in [n_j]$, un element $c_{i1, i2, \ldots, ik}$ defineşte un tablou k dimensional $T(n_1, n_2, \ldots, n_k)$

Secvenţele sunt deci aplicaţii ale domeniului indecşilor pe domeniul de valori.

Deci pentru k = 1 se obţine un tablou unidimensional numit *vector*, $f : [n] \rightarrow C$, având elementele $c_1 = f(1); c_2 = f(2); \ldots ; c_n = f(n)$.

Un tablou este static: nu pot fi inserate sau şterse elemente de tablou (celule). Identificarea elementelor se face cu ajutorul indicilor.

În memoria internă elementele unui tablou static vor ocupa o zonă compactă (contiguă) de memorie, adică locaţii succesive de memorie, ceea ce asigură accesul direct (radom) la un element - reprezentare secvenţială.

Pentru tablourile statice limbajele de programare au un **tip de dată** predefinit:

array în Pascal; **[ ]** C/C+ +,

Accesul la elementul x[i]:
adresa de început a tabloului (a) + poziţia în vector * dimensiunea ocupată în memorie de un element

Tablou Unidimensional:
x: ***array***[n] of TE înseamnă : $x_0,\ldots,x_{n-1}$

x[i]= a + i*sizeof(TE)

x: ***array***[$i_{in}..i_{fin}$] of TE

x[i]= a + (i- i_{in})*sizeof(TE)

Tablou Multidimensional (de k dimensiuni)
reprezentarea în memorie este liniară

La tablourile bidimensionale:
„linie după linie“ (indicii variază de la dreapta la stânga)
„coloană după coloană “ (indicii variază de la stânga la dreapta)

Tablourile k dimensionale - interpretate ca vectori de tablouri k-1 dimensionale.

Accesul la un element $x[i_1,i_2,\ldots,i_{k-1},i_k]$:

x: ***array***$[n_1,n_2,\ldots,n_k]$ of TE

$$x[i_1,i_2,\ldots,i_{k-1},i_k] = a + \left(\sum_{j=1}^{k-1}(i_j - 1) * \prod_{j<i<k+1} n_i\right) * \text{sizeof(TE)}$$

Domeniul indecşilor depinde de limbaj.
Există 3 implementări importante:

index $\in \{0,1,\ldots, n-1\}$

index $\in \{1,\ldots, n\}$ (Matlab,Fortran)

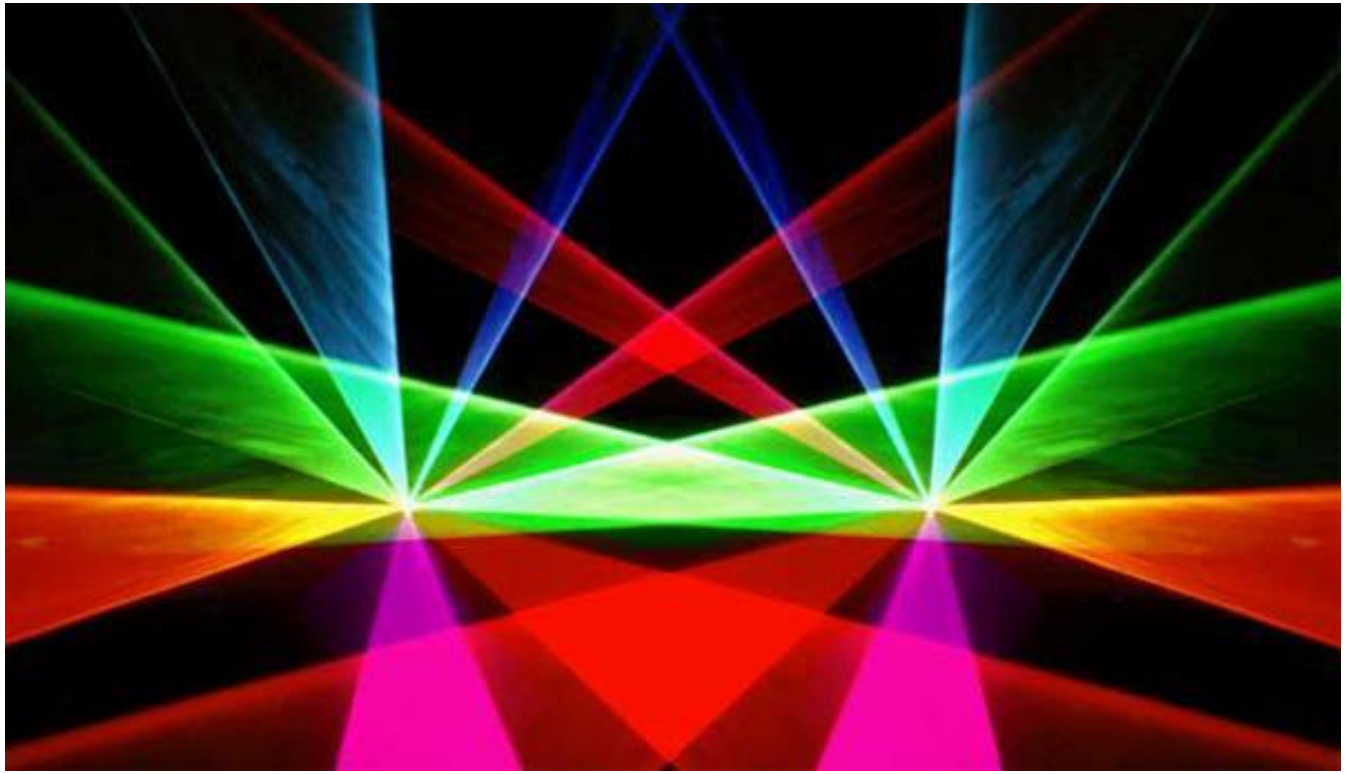
index $\{l_{in},\ldots,l_{fin}\}$

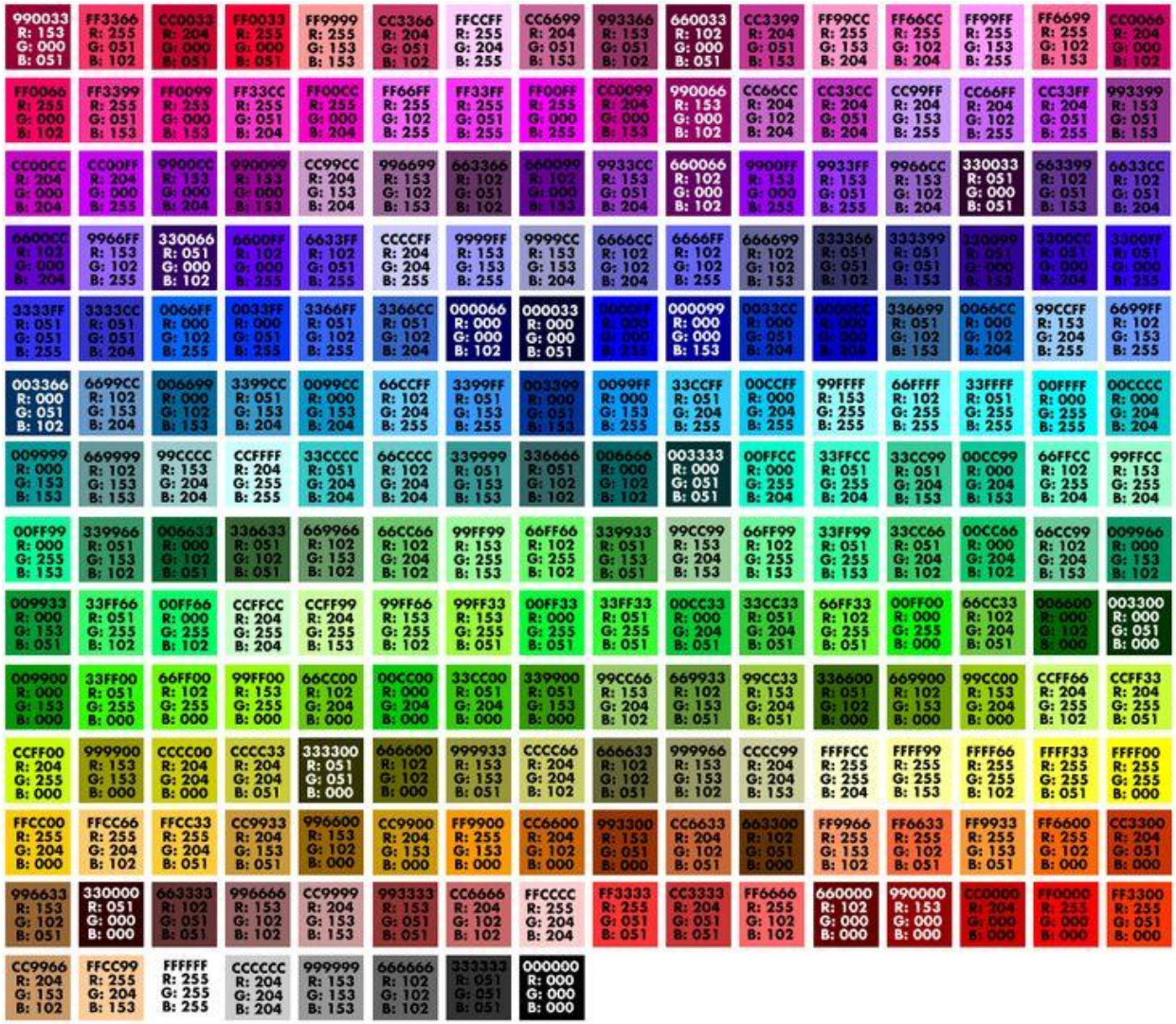
	Default Base index	**Specifiable Index Type**	**Specifiable Base Index**	**Bound Check**	**Multi-dimensional**
C	0	No	No	unchecked	array of array
C++	0	No	No	unchecked	array of array
Java	0	No	No	checked	array of array
Pascal	index type	Yes	Yes	varies	yes

Tablourile k dimensionale interpretate ca vectori de tablouri k-1 dimensionale.

Exemplu:

Imagine RGB 3 dimensiuni
Vector de 3 matrice (una pentru fiecare culoare)





Imagine RGB reprezentare:

- matrice bidimensională de triplete sau
- vector de 3 matrice bidimensionale

Memorarea tablourilor : înlănţuit (cazul matricilor rare)

Tabloul este folosit ca SD pentru mai multe TAD-uri:

- Containere;
- Vectori; Matrice;
- Liste;
- Dicţionare

Exemplu:

Matrice de numere complexe

```
typedef struct {       /*definim tipul elementului */
      double real;
      double imag;
   } complex;
                          /*definim unele operatii*/
complex cadd ( complex z1 , complex z2 )
 {      complex z;
     z.real = z1.real + z2.real;
      z.imag = z1.imag + z2.imag;
      return z;
 }

complex cmul ( complex z1 , complex z2 )
{
 complex z;
 z.real = z1.real * z2.real - z1.imag * z2.imag;
 z.imag = z1.real * z2.imag + z1.imag * z2.real;
return z;
 }
```

```
complex conj ( complex z1 )
 {
  complex z;
 z.real = z1.real;
 z.imag = -z1.imag;
 return z;
 }

 void cprn ( complex z )
 {
     printf("(%lf) + i(%lf)", z.real, z.imag);
 }
```

```
#define MAXROW 10
 #define MAXCOL 15

 typedef struct     /*definim matricea*/
{
 int rowdim;         /* nr. linii*/
 int coldim;          /* nr. coloane*/
 complex entry[MAXROW][MAXCOL];
 } matrix;
```

```
matrix msetid ( int n )       /* matricea unitate*/
 {
 matrix C;
 int i, j;
 if ((n > MAXROW) || (n > MAXCOL)) {
          fprintf(stderr, "msetid: Matrice prea mare\n");
         C.rowdim = C.coldim = 0;
         return C;
   }
  C.rowdim = C.coldim = n;
   for (i = 0; i < C.rowdim; ++i) {
     for (j = 0; j < C.coldim; ++j) {
          A.entry[i][j].real = (i == j) ? 1 : 0;
          A.entry[i][j].imag = 0;
      }
   }
   return C;
 }
```

```
matrix madd ( matrix A , matrix B )       /* adunarea a 2 matrice*/
 {
  matrix C;
  int i, j;

   if ((A.rowdim != B.rowdim) || (A.coldim != B.coldim)) {
       fprintf(stderr, "madd: Matrice de dimensiuni incompatibile\n");
       C.rowdim = C.coldim = 0;
       return C;
    }
    C.rowdim = A.rowdim;
    C.coldim = A.coldim;
    for (i = 0; i < C.rowdim; ++i)
        for (j = 0; j < C.coldim; ++j)
            C.entry[i][j] = cadd(A.entry[i][j],B.entry[i][j]);
    return C;
 }
```

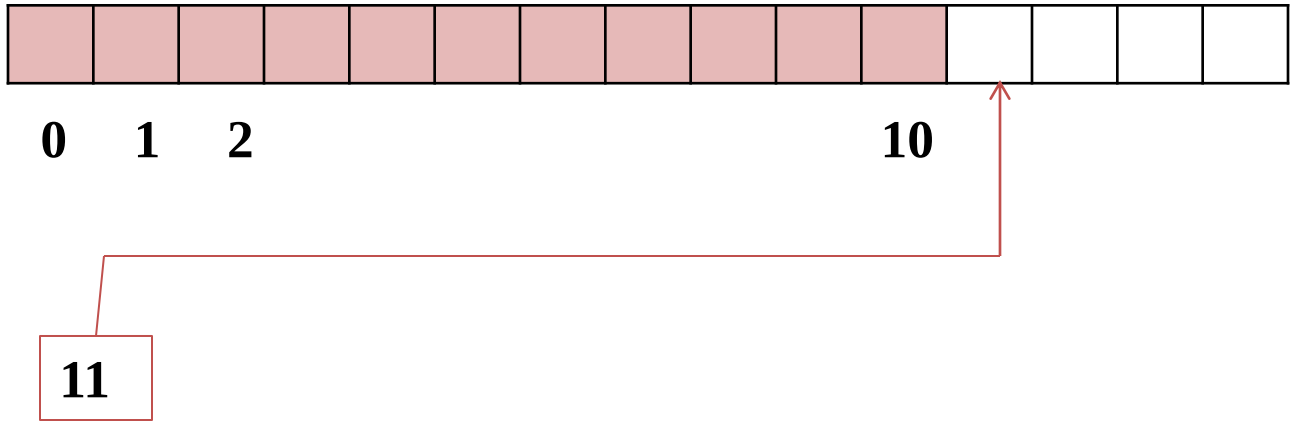
```
matrix mmul ( matrix A , matrix B )      /*inmultirea a 2 matrice*/
 {
   matrix C;
   int i, j, k;
   complex z;

    if (A.coldim != B.rowdim) {
       fprintf(stderr, "mmul: Matrice de dimensiuni incompatibile\n");
        C.rowdim = C.coldim = 0;
        return C;
    }
    C.rowdim = A.rowdim;
    C.coldim = B.coldim;
    for (i = 0; i < A.rowdim; ++i) {
       for (j = 0; j < B.coldim; ++j) {
           C.entry[i][j].real = 0;
           C.entry[i][j].imag = 0;
           for (k = 0; k < A.coldim; ++k) {
              z = cmul(A.entry[i][k], B.entry[k][j]);
              C.entry[i][j] = cadd(C.entry[i][j],z);
            }
         }
      }
      return C;
 }
```

O impementare completa a unuei matrice vezi :

Liviu Negrescu – Limbajele C si C++ pentru incepatori ; Editura Albastra

http://cse.iitkgp.ac.in/~pds/notes/swf/list1.html

Vector dinamic



Constă:

- dintr-un vector (normal) care nu este complet ocupat cu elemente
- dintr-un indice care indică prima poziţie neocupată (primul element liber al vectorului)

Atât timp cât noul element mai încape în vector el va fi inserat => inserarea unui element: *O(1)*
Dacă noul element nu mai încape în câmp (vector) :

- Se generează un nou vector mai mare;
- Se copiază elementele existente în noul vector;
- Se adaugă la sfârşit noul element

Practic:
de câte ori dimensiunea devine insuficientă se generează un vector de dimensiune dublă şi se copiază elementele existente.

În realitate factorul folosit de diferitele limbaje este:

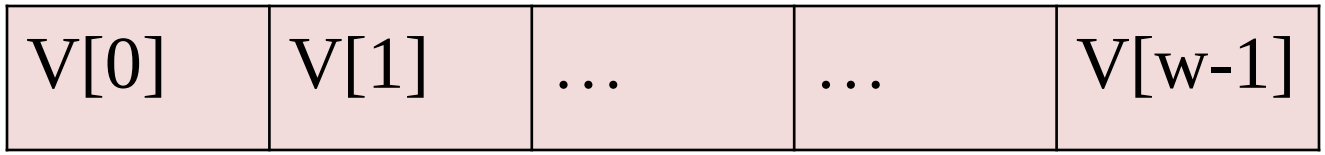
Sun-Java: 1,5
Gnu-Java: 2
C# (Mono): 2
Python: 1,125

Precondiţie:
Afirmaţie care trebuie să fie adevărată înainte de apelul operaţiei (sarcina utilizatorului)

Postcondiţie
Afirmaţie care trebuie să fie adevărată în urma efectuarii operaţiei
=> formează baza argumentării corectitudinii TAD-ului

TAD Vector Dinamic

β = 2 //factorul de creştere
α = 4 //consum max suplimentar de memorie
w = 1 //dimensiunea curentă a câmpului
n = 0 //numărul curent de elemente
v = new TE[w] //vector static



Dăm unele operaţii pentru TAD vector dinamic

Procedure Creare (V,n,TE) **O(n)**
Crează un vector V de n elements de tip TE
Pre : n, TE
Post: V $\in$D

Procedure Set(V,i,e) **O(1)**
Setează e in poziţia i în vectorul V; //de obicei cu []
Pre : V$\in$D, i<n, e $\in$TE
Post: V[i]=e

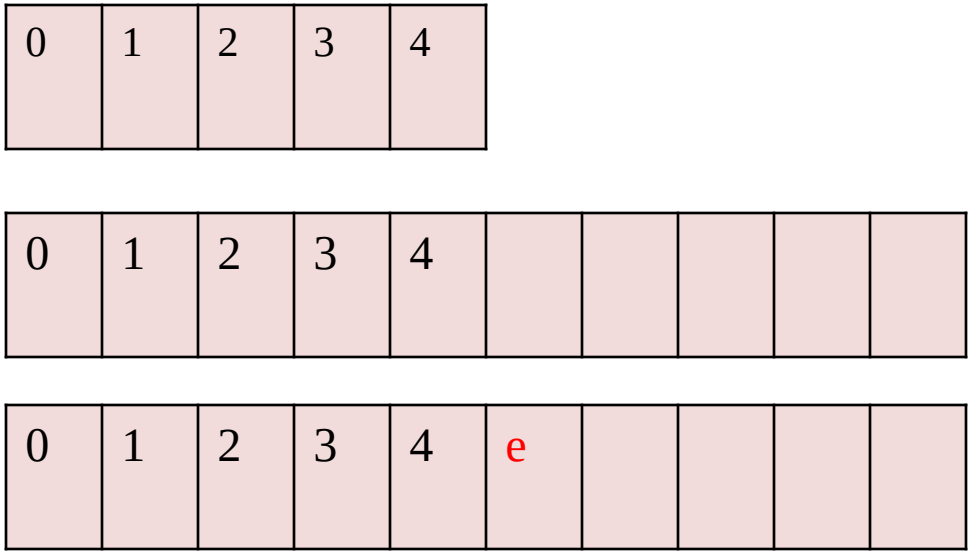
Function Get(V,i) **O(1)**
Returneaza valoarea elementului de pe poziţia i din vectorul v.
Pre : V $\in$D, i <n,
Post: Get = V[i]

Size (dimensiune curentă)

```
set(int i, TE& e) {
 assert(0≤i<n);
v[i] = e;
}

TE get(int i) {
assert(0≤i<n);
return v[i];
}

int size() {
 return n;
}
```

```
void insert (TE e) {
if (n==w)
   reallocate(β*n);
v[n]=e;
n++;
}
```

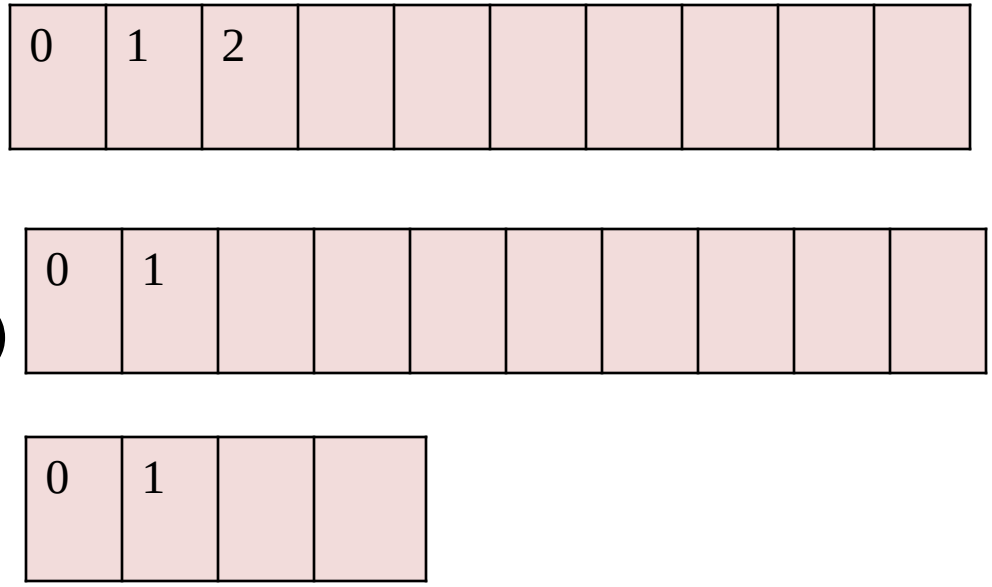
n=5, w=5



n=6, w=10

```
void remove () {
assert(n>0);
n--;
if (α*n ≤w and n>0)
reallocate(β*n);
}
```

n=3, w=10



n=2, w=4

```
void reallocate (int new_w) {
w = new_w;
TE[ ] new_v = new TE[new_w];
for (i=0; i<n; i++)
    new_v[i] = v[i];
v = new_b;
}
```

Foarte curând oamenii se vor împărţi în două categorii: oameni bătrâni şi oameni care ştiu să lucreze la calculator.
(Grigore Moisil)

Grigore Constantin Moisil (n. 10 ianuarie 1906, Tulcea - d. 21 mai 1973, Ottawa, Canada) a fost un matematician român, considerat părintele informaticii românești cu invenția de circuite electronice tristabile