Mutatók és vektorok kapcsolata

Kormányos Andor

Komplex Rendszerek Fizikája Tanszék

2022. szeptember 27.

Mutatók és vektorok

A C-ben a mutatók és a vektorok között szoros kapcsolat van. Minden művelet, ami egy tömb indexelésével elvégezhető, megoldható mutatókkal is.

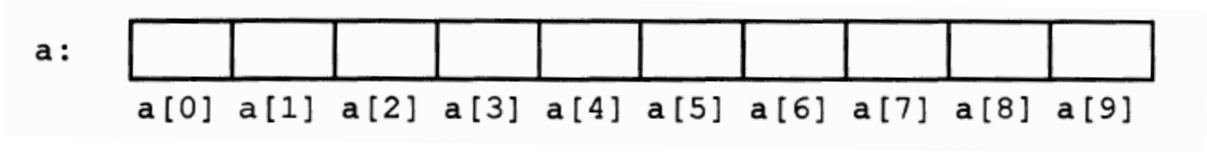
Mutatók és vektorok

A C-ben a mutatók és a vektorok között szoros kapcsolat van. Minden művelet, ami egy tömb indexelésével elvégezhető, megoldható mutatókkal is.

Definiáljunk egy 10 elemű vektort:

```
double a[10];
```

A vektor elemei egy adott címtől kezdve folytonosan helyezkednek el a memóriában



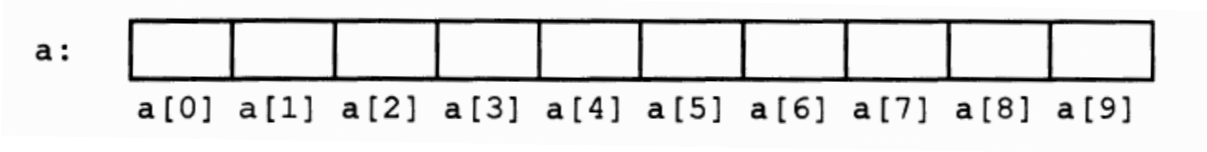
Mutatók és vektorok

A C-ben a mutatók és a vektorok között szoros kapcsolat van. Minden művelet, ami egy tömb indexelésével elvégezhető, megoldható mutatókkal is.

Definiáljunk egy 10 elemű vektort:

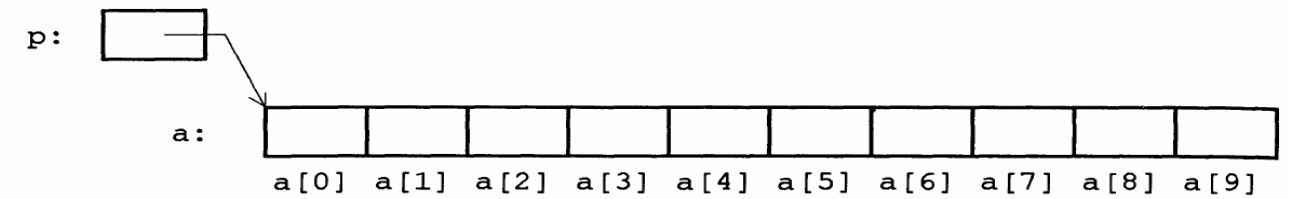
```
double a[10];
```

A vektor elemei egy adott címtől kezdve folytonosan helyezkednek el a memóriában



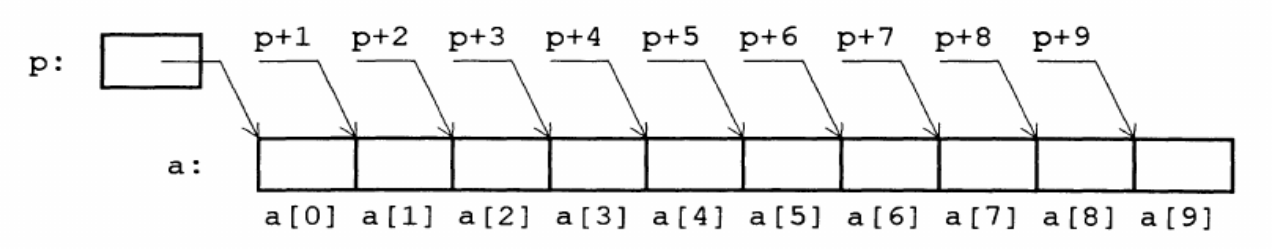
Definiálhatunk egy pointert és ráállíthatjuk az **a** vektor 0-ik elemének címére:

```
1  double a[10];
2  double *p;
3
4  p = &a[0];
5  // p most az a tomb 0-ik elemere mutat
```



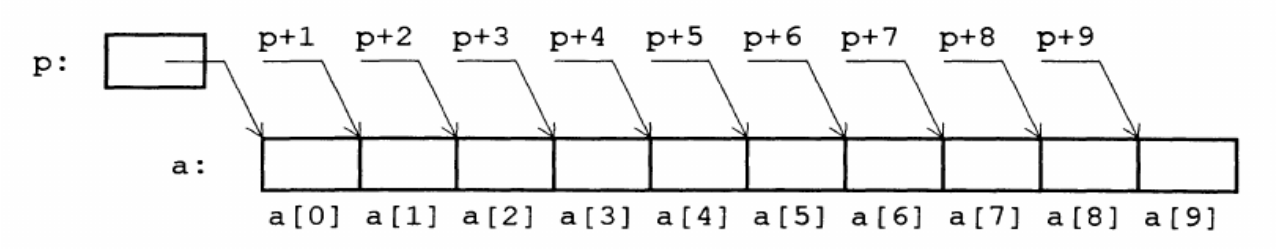
Mutatók és vektorok

- ha a p= &a[0] paranccsal az **a** vektor 0-ik elemének címére állítottuk a *p* mutatót ⇒ p+i kifejezéssel az **a** vektor *i*-ik elemének címére mutatunk



Mutatók és vektorok

- ha a p= &a[0] paranccsal az **a** vektor 0-ik elemének címére állítottuk a *p* mutatót ⇒ p+i kifejezéssel az **a** vektor *i*-ik elemének címére mutatunk



- tömb neve és 0-ik indexű elemének a címe igazából szinonímák, ezért

```
1  double a[10];
2  double *p;
3
4  p = &a[0];
```

mellett használhatjuk

```
1  double a[10];
2  double *p;
3
4  p = a;    // p most az a tomb 0-ik elemere mutat
```

Miután a p pointert ráállítottuk egy **a** vektor első elemének a címére, a következők egyenértékűek:

`*(p + i)`	$\Leftrightarrow$	`p[i]`	az i-ik elem értéke
`p + i`	$\Leftrightarrow$	`&p[i]`	az i-ik elem címe

Fontos: a *p* mutató egy változó, értéke változtatható, az **a** vektor viszont egy konstans tömb.

```
1  double a[10];
2  double *p;
3
4  p = &a[0];
5
6  p[5]=p[5]+1; // miutan p-t raallitottuk a vektor elso elemere
7               // a vektor i-ik  elemenek ertekere  p[i]-vel is
8               // hivatkozhatunk
9
10  *(p++)=5;  // a mutato most a[1] cimere mutat es erteke 5
11
12 (*p)++;  // a mutato altal kijelolt memoriacimen levo
13         // erteket megnoveltuk
14
15
16 a++;  // helytelen, hibauzenetet kapunk
17
18 a[5]++;  // helyes, az a vektor 5-ik elemet noveltuk
```

A pointer és a mutatott típus mérete

- a pointer memóriacímre mutat, de tisztában van a mutatott típus méretével
- ezt figyelembe veszi, amikor a következő elem memóriacímét kérdezzük le
- a mutatott típus méretétől függ, hogy hány byte-t ugrik