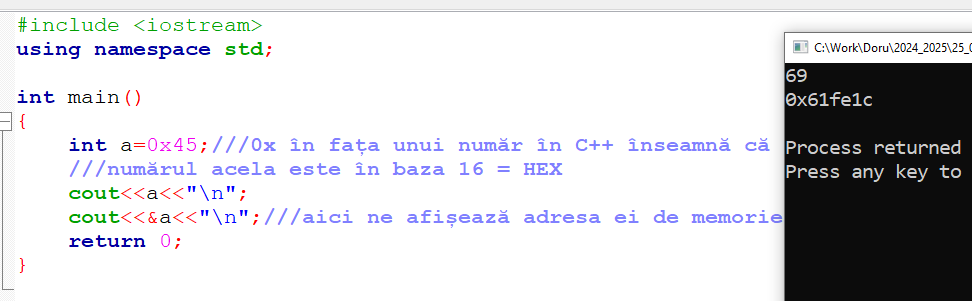
**Variabile statice și variabile dinamice (pointeri)**

Toate variabilele cu care am lucrat până acum sunt variabile statice.

Ele au o adresă de memorie FIXĂ de la lansarea în execuție a programului până la terminarea sa.

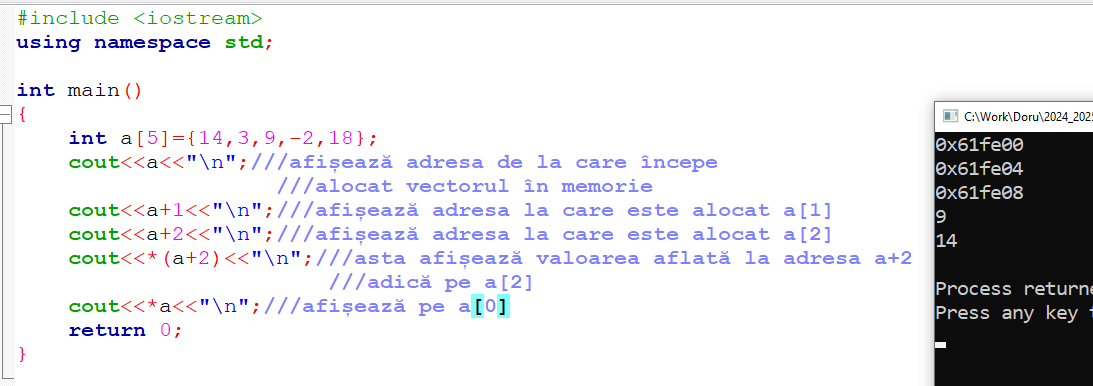
Această adresă este neinteresantă d.p.d.v. al utilizatorului (ea fiind un număr de ordine al octetului care găzduiește începutul reprezentării variabilei) dar totuși poate fi obținută cu operatorul &.

Chiar și vectorii sunt de asemenea statici.

În cazul lor, numele variabilei FĂRĂ paranteze drept reprezintă ADRESA de memorie de la care începe să fie alocat vectorul.

Adunările cu o constantă **x** la această adresă înseamnă adresa de memorie a elementului de la indicele **x**.

În cazul vectorilor - dacă lucrăm cu adrese de memorie, operatorul \* ne dă de fapt valoarea care se află la adresa respectivă.



De reținut așadar:

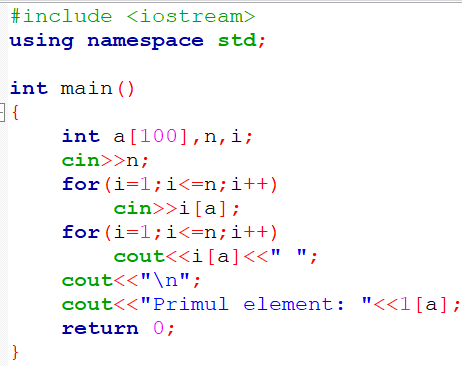
Dacă a este un vector, expresia **\*a** <=> **a[0]**

expresia **\*(a+x)** <=> **a[x]**

Funny fact:

As we saw: **a[x]=\*(a+x)**

adunarea este comutativă deci putem continua **=\*(x+a)=x[a]**



**Pointeri (sau variabile dinamice)**

Sunt variabilele care se declară cu \*.

Ex:

**int \*a;**

Imediat după declarare, aceste variabile sunt homeless - NU au alocată NICIO adresă de memorie.

Putem face două lucruri cu un pointer nealocat:

1) îi alocăm memorie prin:

**a=new int;** ///exemplul este pentru o simplă variabilă de tip int

sau

**a=new int[nr\_elemente];**///așa se alocă un vector cu **nr\_elemente**

În primul caz (variabila simplă), adresarea variabilei a se poate face prin:

a = adresa de memorie

\*a = valoarea aflată la acea adresă de memorie

În al doilea caz (vectorul) lucrăm cu a exact așa cum știam deja de la vectori, adică putem folosi a[indice]

2) Îl ducem să "paraziteze" o altă variabilă (o adresă de memorie)