Structuri de date – Curs 2

Prof. univ. dr. Cristian CIUREA
Departamentul de Informatică și Cibernetică Economică
Academia de Studii Economice din București
cristian.ciurea@ie.ase.ro

Agenda

- Memorie Stack/Heap
- Clase de memorie ale variabilelor
- Tipuri de date
- Lista liniară
- Lista simplă liniară
- Lista dublă liniară
- Bibliografie

Memoria Stack

- Utilizată de variabile locale – definite în funcții.
- Utilizată pentru transfer parametri în funcții.
- Localizată la capătul memoriei accesibile => alocare descrescătoare a adreselor.
- Utilizată pentru variabilele locale, valori temporare, argumente ale funcțiilor, adrese de return.
- Implementată prin structura de date de tip stivă.
- Dimensiune cunoscută la momentul compilării aplicației și alocată la momentul de începere a execuției aplicației.

Memoria Heap

- Alocare memorie pe durata de execuție a aplicației.
- Alocare gestionată de sistemul de operare.
- Zone de memorie gestionate prin variabile de tip pointer.
- Memorie alocată dinamic (pe dimensiunea cunoscută la execuția aplicației).
- Accesată prin pointeri și are un conținut anonim;

Memoria Heap

- Dimensiune până la pointerul ***break***. Poate fi modificată (adăugare) prin cerere către sistemul de operare;
- Posibilitate de creștere a dimensiunii prin mutarea pointerului ***break*** spre adrese mai mari;
- Fragmentare ridicată prin operații multiple de alocare / dezalocare.
- Responsabilitate dezalocare (memory leaks).

Clase de memorie ale variabilelor

AUTOMATICE (specificator **auto)**:

- Locale pentru blocul de instrucțiuni în care se definesc variabilele.
- Persistente până la terminarea blocului de instrucțiuni în care se definesc.
- Zone de memorie distincte pentru cod recursiv sau multi-threading.
- Stocate în segmentul de stivă;

Exemple:

```
auto a = 7;
auto b = "VariabileAuto";
```

Clase de memorie ale variabilelor

REGISTRU (specificator **register)**:

- Specificator utilizat doar pentru variabile locale și parametri ai funcțiilor.
- Persistente până la terminarea blocului de instrucțiuni în care se definesc (similar specificatorului auto).
- Decizia compilatorului de încărcare a unui registru cu conținut variabilă.
- Utile pentru operații frecvente din punctul de vedere al reducerii timpului de acces și execuției;

Exemplu:

```
register int vreg;
int d;
d = 8;
vreg = d;
```

Clase de memorie ale variabilelor

EXTERNE (specificator **extern**):

- Utilizate pentru variabile declarate în mai multe fișiere sursă.
- Memorie alocată înainte de execuția funcției main(); persistentă până la terminarea execuției programului.
- Definite în bloc de instrucțiuni cu accesibilitate în cadrul blocului; altfel, accesibilă la nivel de fișier sursă;

Exemplu:

Fisierul 1

```
extern int i;
void f() {
  i++;
}
```

Fisierul 2

```
int i = 0;
extern void f();
void g() {
  f();
  printf("%d\n", i);
}
```

Clase de memorie ale variabilelor

STATICE (specificator **static**):

- Persistență conținut și vizibilitate în blocul unde sunt definite (chiar și la nivel de fișier). La nivel de fișier individual, pot fi asimilate variabilelor globale, dar vizibilitatea este diferită într-o colecție de fișiere sursă (abordare static vs. global).
- Alocate la începerea execuției programului și dezalocate la terminare execuției.
- Declararea într-o funcție asigură persistența conținutului între apeluri.

Exemplu:

```
int f() {
  static int x = 0;
  x++;
  return x;
}
void main() {
  int j;
  for (j = 0; j < 10; j++) {
     printf("Rezultat functie f: %d\n", f());
  }
}
```

Tipuri de date

Sursa definirii:

- Fundamentale, definite în cadrul limbajului;
- Definite de utilizator (programator/dezvoltator etc); exemple: structuri articol, clase de obiecte, structuri pe biți, uniuni etc.

Natura conținutului:

- Simple, corespunzătoare tipului de date.
- Masive, agregate și accesibile prin indecși.
- Pointeri, acces explicit la zone de memorie.

Tipuri de date

Denumire	Explicatie	Dimensiune în bytes	Interval valori posibile
char	Caracter sau întreg de valoare mica.	1 byte	cu semn: -128 la 127 fara semn: 0 la 255
short int (short)	Întreg short.	2 bytes	cu semn: -32768 la 32767 fara semn: 0 la 65535
int	Întreg.	4 bytes	cu semn: -2147483648 la 2147483647 fara semn: 0 la 4294967295
long int (long)	Întreg long.	4 bytes	cu semn: -2147483648 la 2147483647 fara semn: 0 to 4294967295
float	Real virgulă mobilă precizie simpla.	4 bytes	+/- 3.4e +/- 38
double	Real virgulă mobilă precizie dublă.	8 bytes	+/- 1.7e +/- 308
long double	Real virgulă mobilă precizie extinsă.	8 bytes / 10 bytes / 16 bytes (functie de compilator)	

Tipuri de date

Alocare dinamică memorie:

- Funcții: **malloc()**
- Rezervă memorie în **HEAP**

Dezalocare memorie:

- Funcție: **free()**
- Eliberează memoria rezervată în **HEAP**

Lista liniară

Lista liniară:

- colecție de elemente denumite noduri;
- relație de ordine rezultată din poziția nodurilor;
- elemente de același tip – structură omogenă;
- lungimea listei – numărul de noduri;
- asemănătoare structurii de tip masiv unidimensional;
- structură alocată în HEAP – gestionată prin variabila pointer.

Lista liniară

Lista liniară	Vector
Număr variabil de elemente, fără declararea anticipată a dimensiunii	Număr predeterminat de elemente, cu rezervarea de memorie pe dimensiunea declarată
Elemente alocate la adrese nu neapărat consecutive	Spațiu contiguu de memorie (elementele sunt adiacente logic si fizic)
Alocare memorie la execuție	Alocare memorie la compilare/ execuție
Referire secvențială a nodurilor pornind de la adresa primului nod	Referire prin deplasament al elementului față de adresa de început a zonei

Lista liniară

Implementare:

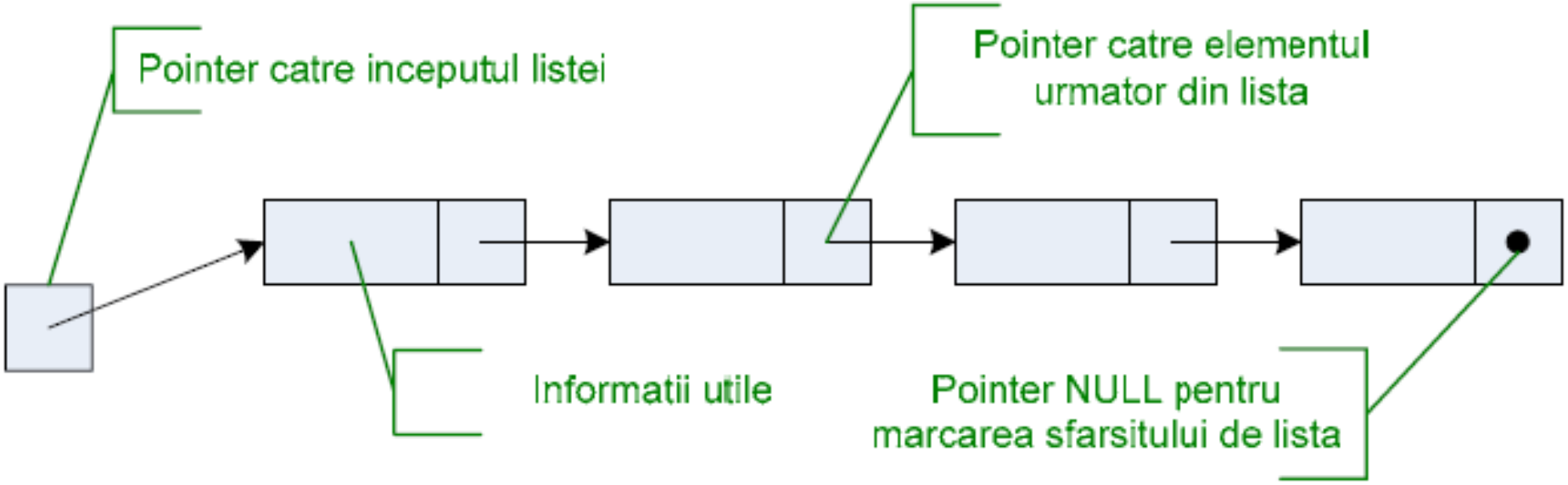
- utilizare variabile pointer;
- nodurile conțin două categorii de informații:
  - câmpuri cu informația structurală (utilă);
  - câmpuri cu informația de legătură – adrese ale altor noduri din cadrul listei.

Lista simplă

Lista simplă / simplu înlănțuită:

- listă liniară;
- structură de date dinamică omogenă;
- noduri alocate ca elemente separate de memorie;
- un singur câmp cu informația de legătură – nodul succesor al câmpului curent;
- ultimul nod nu are succesor – informația de legătură este nulă;
- gestionată prin variabila pointer cu adresa primului nod din listă;
- permite numai acces secvențial la elemente.

Lista simplă



Lista simplă

Lista simplă:

- definirea structurii unui nod dintr-o listă simplă:

```
typedef struct {
        char inf;
        struct Nod *next;
} Nod;
```

- declararea unei liste simple vide:

```
Nod *prim = NULL;
```

- alocarea de memorie heap pentru un nod al listei simple:

```
Nod *nou = (Nod*)malloc(sizeof(Nod));
```

Lista simplă

Lista simplă:

- dezalocarea de memorie heap pentru un nod al listei simple:

```
free(nou);
```

- inițializarea și referirea informației utile dintr-un nod al listei simple:

```
nou->inf='A';
```

- inițializarea și referirea informației de legătură dintr-un nod al listei simple:

```
nou->next = NULL;
```

Lista simplă

Operații cu liste simple:

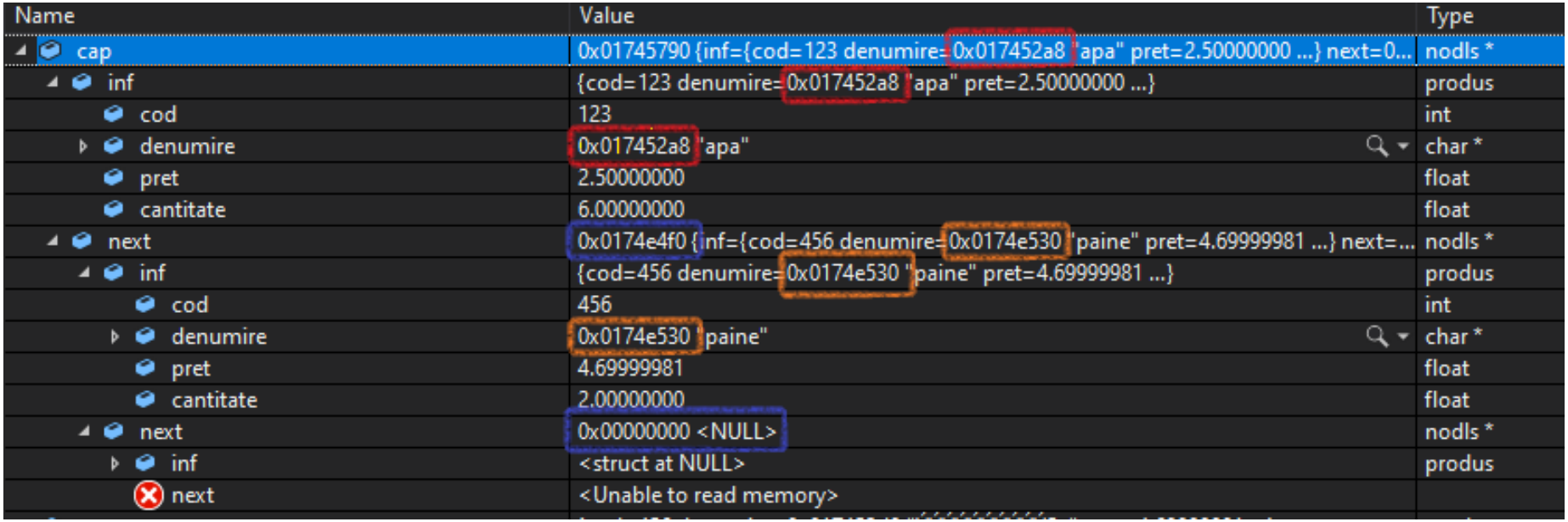
- Crearea listei;
- Inserare nod;
- Traversare;
- Căutare;
- Interschimbare noduri;
- Ștergere nod;
- Sortarea listei;
- Concatenarea de liste;
- Interclasare liste;

Lista simplă

```
typedef struct
{
        int cod;
        char* denumire;
        float pret;
        float cantitate;
} produs;

typedef struct
{
        produs inf;
        struct nodls* next;
} nodls;
```

Lista simplă



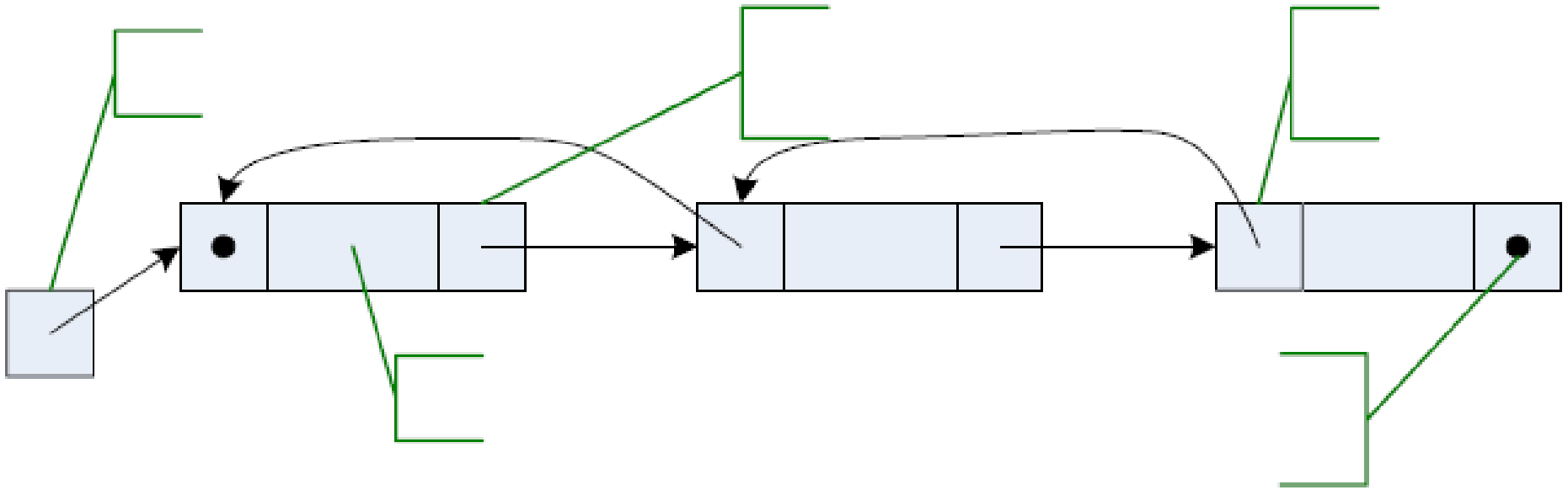
Name	Value	Type
cap	0x01745790 {inf={cod=123 denumire=0x017452a8 "apa" pret=2.50000000 ...} next=0...	nodls *
inf	{cod=123 denumire=0x017452a8 "apa" pret=2.50000000 ...}	produs
cod	123	int
denumire	0x017452a8 "apa"	char *
pret	2.50000000	float
cantitate	6.00000000	float
next	0x0174e4f0 {inf={cod=456 denumire=0x0174e530 "paine" pret=4.69999981 ...} next=...	nodls *
inf	{cod=456 denumire=0x0174e530 "paine" pret=4.69999981 ...}	produs
cod	456	int
denumire	0x0174e530 "paine"	char *
pret	4.69999981	float
cantitate	2.00000000	float
next	0x00000000 <NULL>	nodls *
inf	<struct at NULL>	produs
next	<Unable to read memory>	

Lista dublă

Lista dublă / dublu înlănțuită:

- listă liniară;
- structură de date dinamică omogenă;
- noduri alocate ca elemente separate de memorie;
- două câmpuri cu informații de legătură – nodurile succesor, respectiv predecesor ale câmpului curent;
- ultimul nod nu are succesor – informația de legătură este nulă;
- primul nod nu are predecesor – informația de legătură este nulă;
- permite traversarea în ambele direcții.

Lista dublă



Lista dublă

Lista dublă:

- definirea structurii unui nod dintr-o listă dublă:

```
typedef struct {
        char inf;
        struct NodD *prev, *next;
} NodD;
```

- declararea unei liste duble vide:

```
ListaD lst;
lst.prim=NULL;
lst.ultim=NULL;
```

- alocarea de memorie heap pentru un nod al listei duble:

```
NodD *nou = (NodD*)malloc(sizeof(NodD));
```

Lista dublă

Lista dublă:

- dezalocarea de memorie heap pentru un nod al listei duble:

```
free(nou);
```

- inițializarea și referirea informației utile dintr-un nod al listei duble:

```
nou->inf='Z';
```

- inițializarea și referirea informațiilor de legătură dintr-un nod al listei duble:

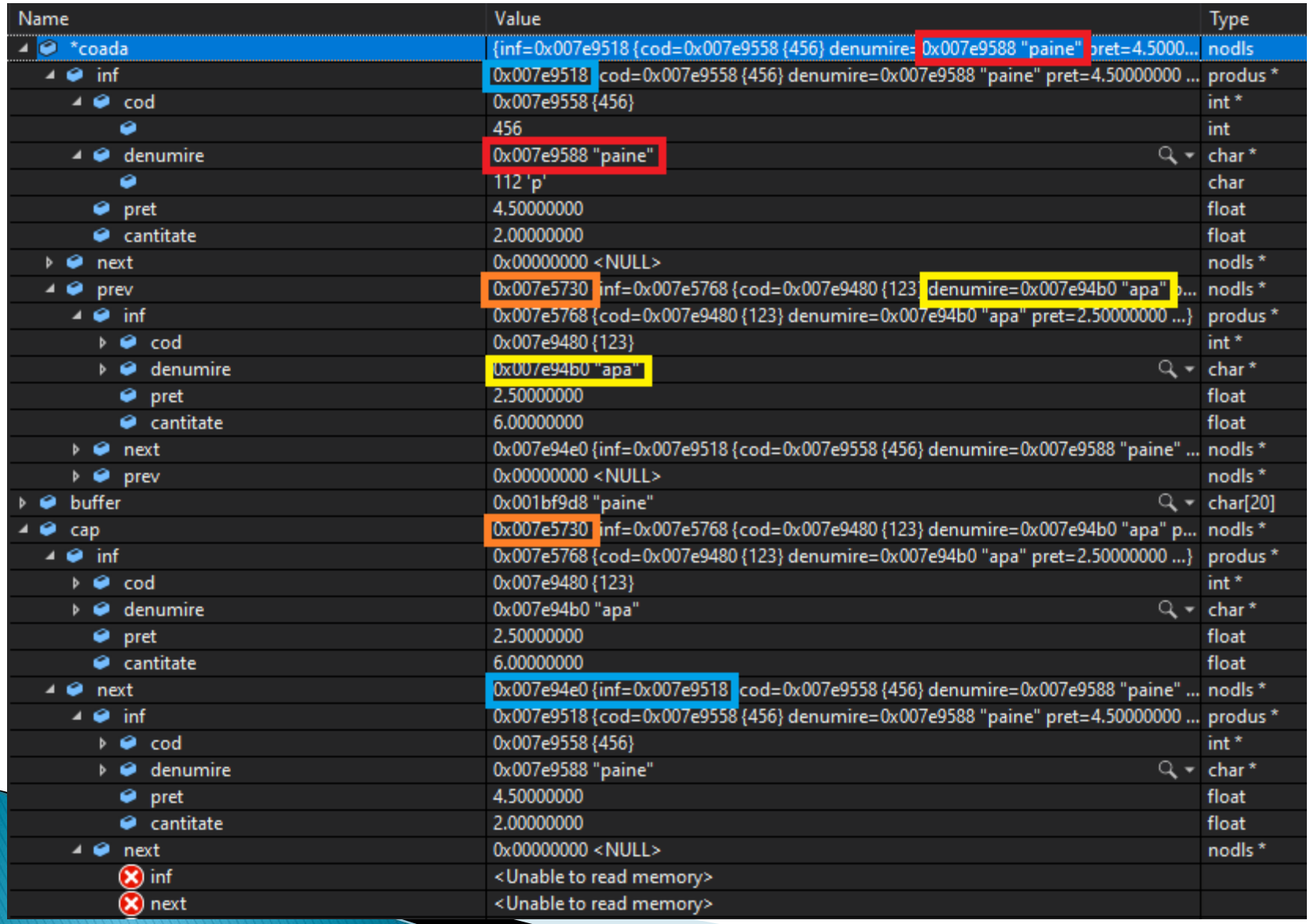
```
nou->next=NULL;
nou->prev=NULL;
```

Lista dublă

```
typedef struct
{
        int* cod;
        char* denumire;
        float pret;
        float cantitate;
} produs;

typedef struct nodls
{
        produs* inf;
        struct nodls* next, * prev;
};
```

Lista dublă



Name	Value	Type
*coada	{inf=0x007e9518 {cod=0x007e9558 {456} denumire= 0x007e9588 "paine" pret=4.5000...	nodls
inf	0x007e9518 cod=0x007e9558 {456} denumire=0x007e9588 "paine" pret=4.50000000 ...	produs *
cod	0x007e9558 {456}	int *
	456	int
denumire	0x007e9588 "paine"	char *
	112 'p'	char
pret	4.50000000	float
cantitate	2.00000000	float
next	0x00000000 <NULL>	nodls *
prev	0x007e5730 inf=0x007e5768 {cod=0x007e9480 {123} denumire=0x007e94b0 "apa" ...	nodls *
inf	0x007e5768 {cod=0x007e9480 {123} denumire=0x007e94b0 "apa" pret=2.50000000 ...}	produs *
cod	0x007e9480 {123}	int *
denumire	0x007e94b0 "apa"	char *
pret	2.50000000	float
cantitate	6.00000000	float
next	0x007e94e0 {inf=0x007e9518 {cod=0x007e9558 {456} denumire=0x007e9588 "paine" ...	nodls *
prev	0x00000000 <NULL>	nodls *
buffer	0x001bf9d8 "paine"	char[20]
cap	0x007e5730 inf=0x007e5768 {cod=0x007e9480 {123} denumire=0x007e94b0 "apa" p...	nodls *
inf	0x007e5768 {cod=0x007e9480 {123} denumire=0x007e94b0 "apa" pret=2.50000000 ...}	produs *
cod	0x007e9480 {123}	int *
denumire	0x007e94b0 "apa"	char *
pret	2.50000000	float
cantitate	6.00000000	float
next	0x007e94e0 {inf=0x007e9518 cod=0x007e9558 {456} denumire=0x007e9588 "paine" ...	nodls *
inf	0x007e9518 {cod=0x007e9558 {456} denumire=0x007e9588 "paine" pret=4.50000000 ...	produs *
cod	0x007e9558 {456}	int *
denumire	0x007e9588 "paine"	char *
pret	4.50000000	float
cantitate	2.00000000	float
next	0x00000000 <NULL>	nodls *
inf	<Unable to read memory>	
next	<Unable to read memory>	

Bibliografie

- Marius Popa, Cristian Ciurea, Mihai Doinea, Alin Zamfiroiu – *Structuri de date: teorie şi practică*, Editura ASE, Bucureşti, 2023, 280 pg.
  - Cap. 4. Structuri dinamice liniare înlănțuite