

# STRUCTURI DE DATE

**Lista simpla**  
**Lista dubla**

# STRUCTURI LINIARE DINAMICE

Structura liniara dinamica:

- Colectie de elemente denumite noduri.
- Relatie de ordine rezultata din pozitia nodurilor.
- Elemente de acelasi tip – structura omogena.
- Lungimea listei – numarul de noduri.
- Asemnatoare structurii de tip masiv unidimensional.
- Structura alocata in heap – gestionata prin variabila pointer.

# STRUCTURI LINIARE DINAMICE

Structura liniara dinamica vs. Vector:

Lista liniara	Vector
Numar variabil de elemente, fara declararea anticipata a dimensiunii	Numar predeterminat de elemente, cu rezervarea de memorie pe dimensiunea declarata
Elemente alocate la adrese nu neaparat consecutive	Spatiu contiguu de memorie (elementele sunt adiacente logic si fizic)
Alocare memorie la executie	Alocare memorie la compilare/executie
Referire secventiala a nodurilor pornind de la adresa primului nod	Referire prin deplasament al elementului fata de adresa de inceput a zonei

# STRUCTURI LINIARE DINAMICE

Implementare:

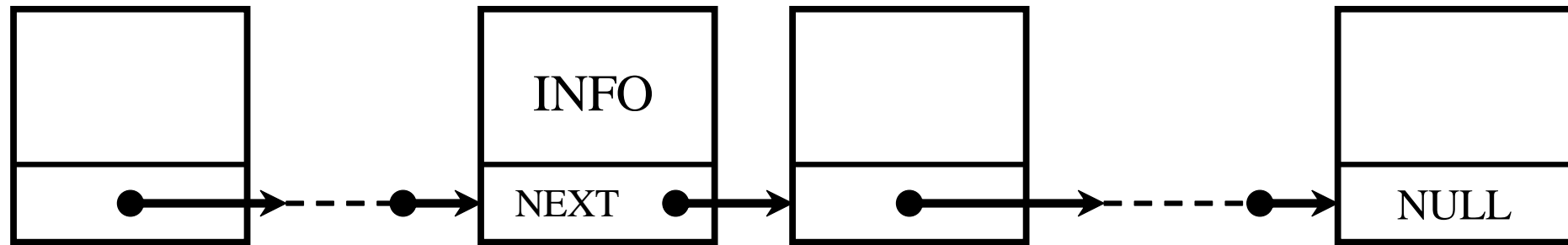
- Utilizare variabile pointer.
- Nodurile contin doua categorii de date:
  - Campuri cu date structural.
  - Campuri cu date de legatura – adrese ale altor noduri din cadrul listei.

# LISTA SIMPLA

Lista simpla:

- Structura liniara dinamica.
- Un singur camp cu informatia de legatura – nodul succesori al campului current.
- Ultimul nod nu are succesori – informatia de legatura este pointer nul (NULL sau 0).
- Gestionata prin variabila pointer cu adresa primului nod din lista liniara.

# LISTA SIMPLA



Definirea structurii unui nod dintr-o lista simpla:

```
struct Nod {  
    char inf; // date de prelucrat  
    Nod* next; // date de legatura; acces la nod succesor  
};
```

# LISTA SIMPLA

Declararea unei liste simple vide:

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

Alocarea de memorie heap pentru un nod al liste simple:

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

Dezalocarea de memorie heap pentru un nod al liste simple:

```
free(nou);
```

# LISTA SIMPLA

Initializarea si referirea informatiei utile dintr-un nod al listei simple:

```
nou->inf = 'A';  
(*nou).inf
```

Initializarea si referirea informatiei de legatura dintr-un nod al listei simple:

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



# LISTA SIMPLA

Operatii cu liste simple:

Creare listei

Inserare nod

Traversare

Cautare

Interschimbare noduri

Stergere nod

Stergerea listei

Sortarea listei

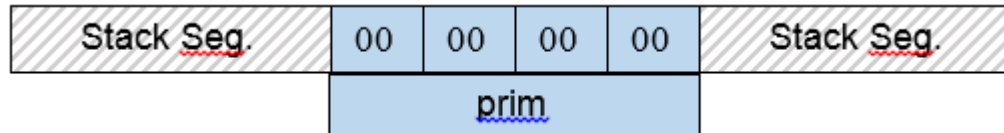
Concatenarea de liste

Interclasare liste.

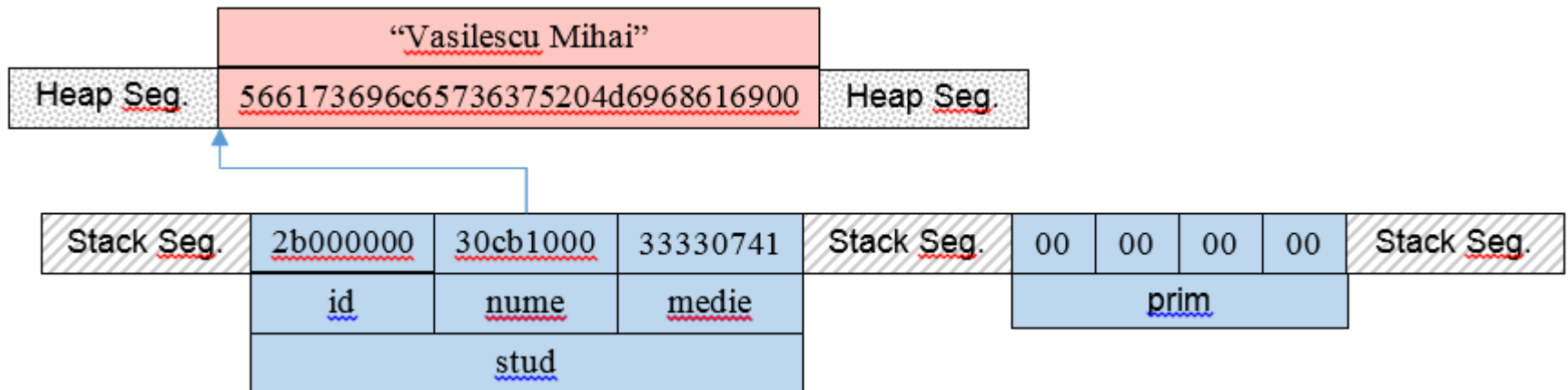
# LISTA SIMPLA

Creare listei

```
Nod *prim = 0;
```

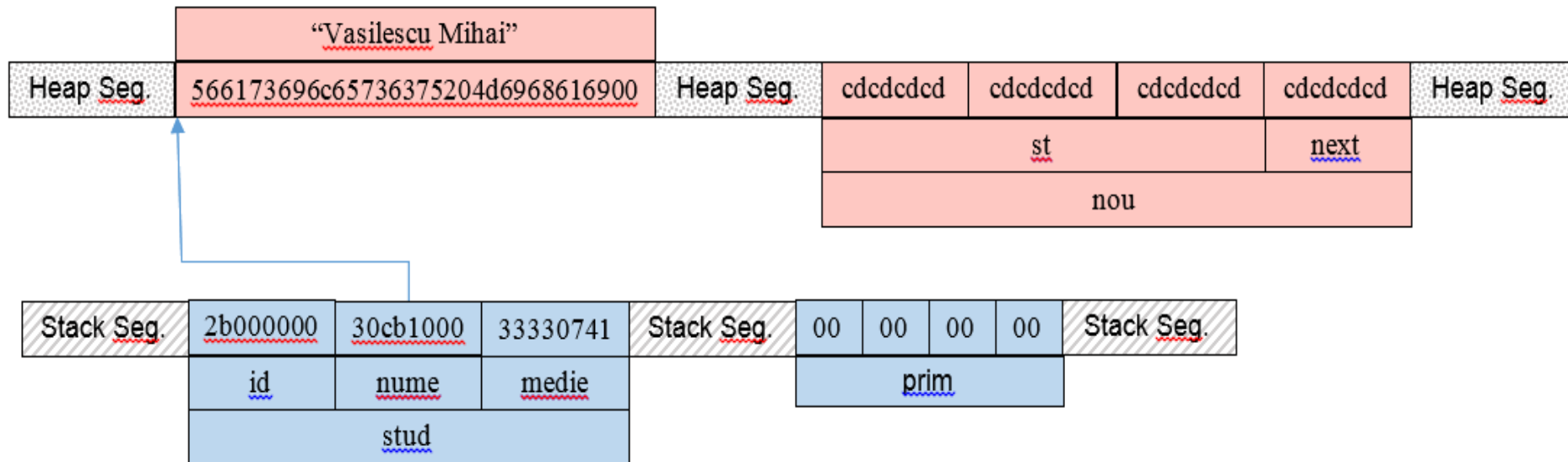


```
stud.id = atoi(token); // 43  
stud.num = (char*)malloc((strlen(token) + 1) * sizeof(char));  
strcpy(stud.num, token); // Vasilescu Mihai  
stud.medie = atof(token); // 8.45
```



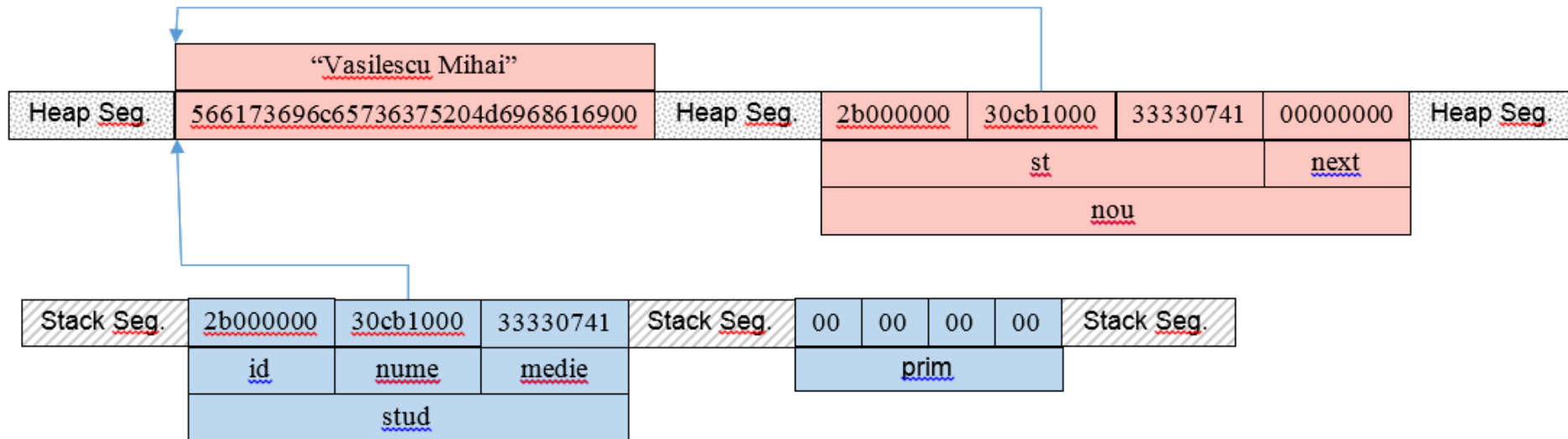
# LISTA SIMPLA

```
Nod* nou;  
nou = (Nod*)malloc(1 * sizeof(Nod));
```



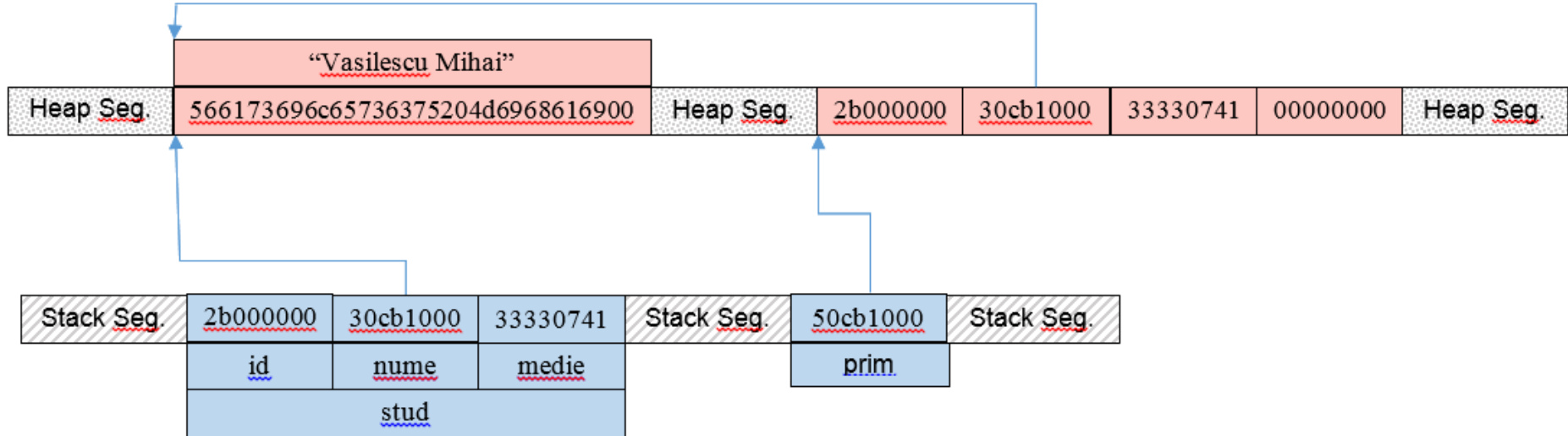
# LISTA SIMPLA

```
nou->st = stud;  
nou->next = prim;
```



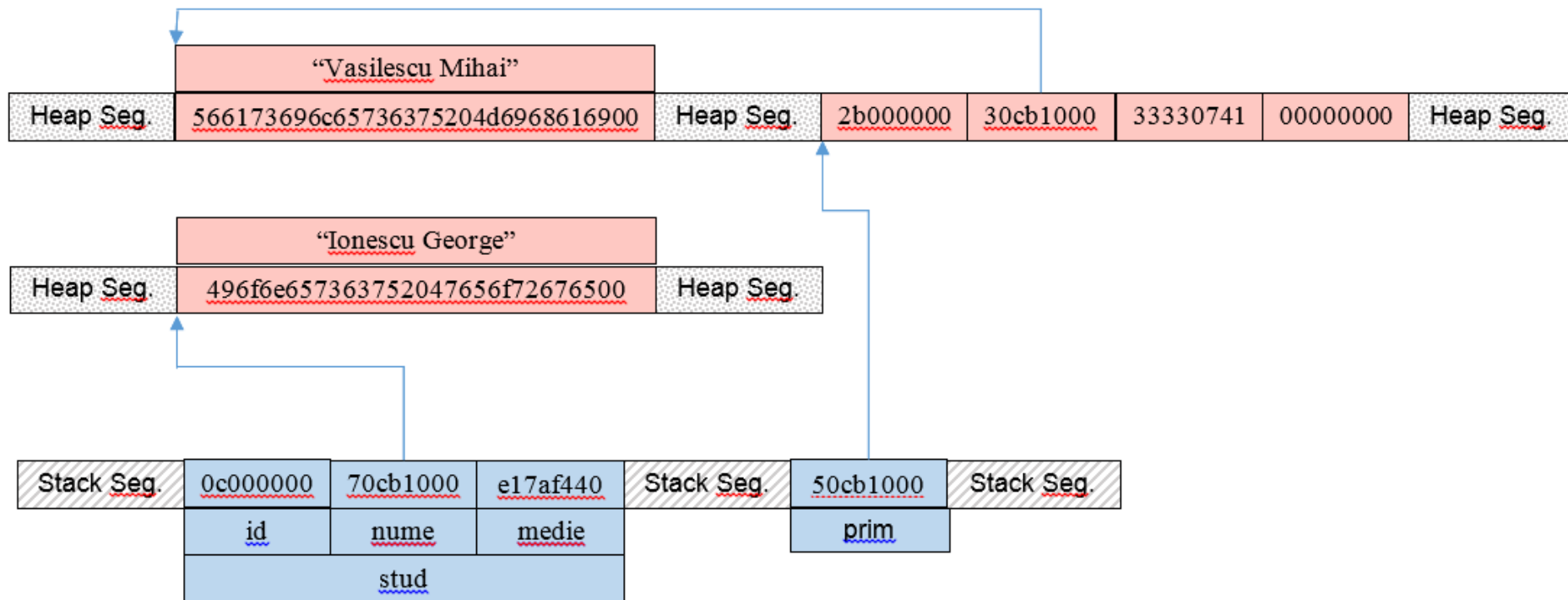
# LISTA SIMPLA

```
prim = inserareLista(prim, stud);
```



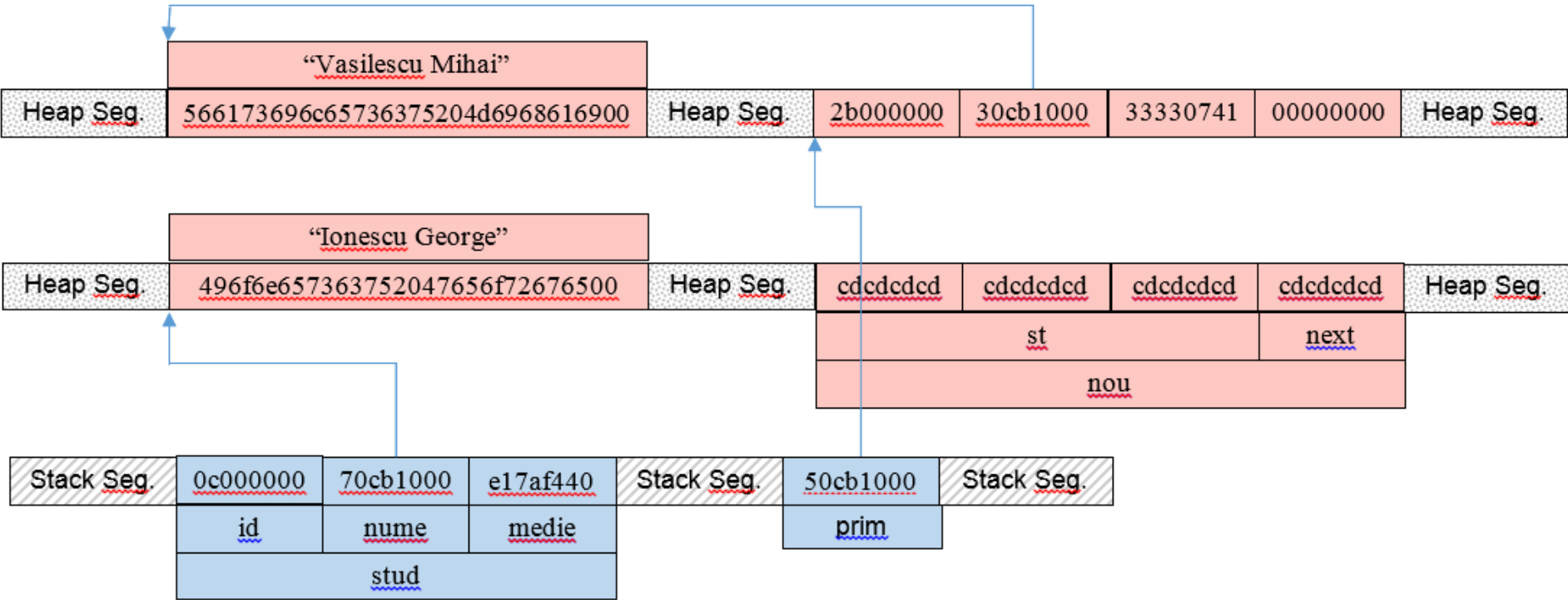
# LISTA SIMPLA

```
stud.id = atoi(token); // 12
stud.nume = (char*)malloc((strlen(token) + 1) * sizeof(char));
strcpy(stud.nume, token); // Ionescu George
stud.medie = atof(token); // 7.64
```



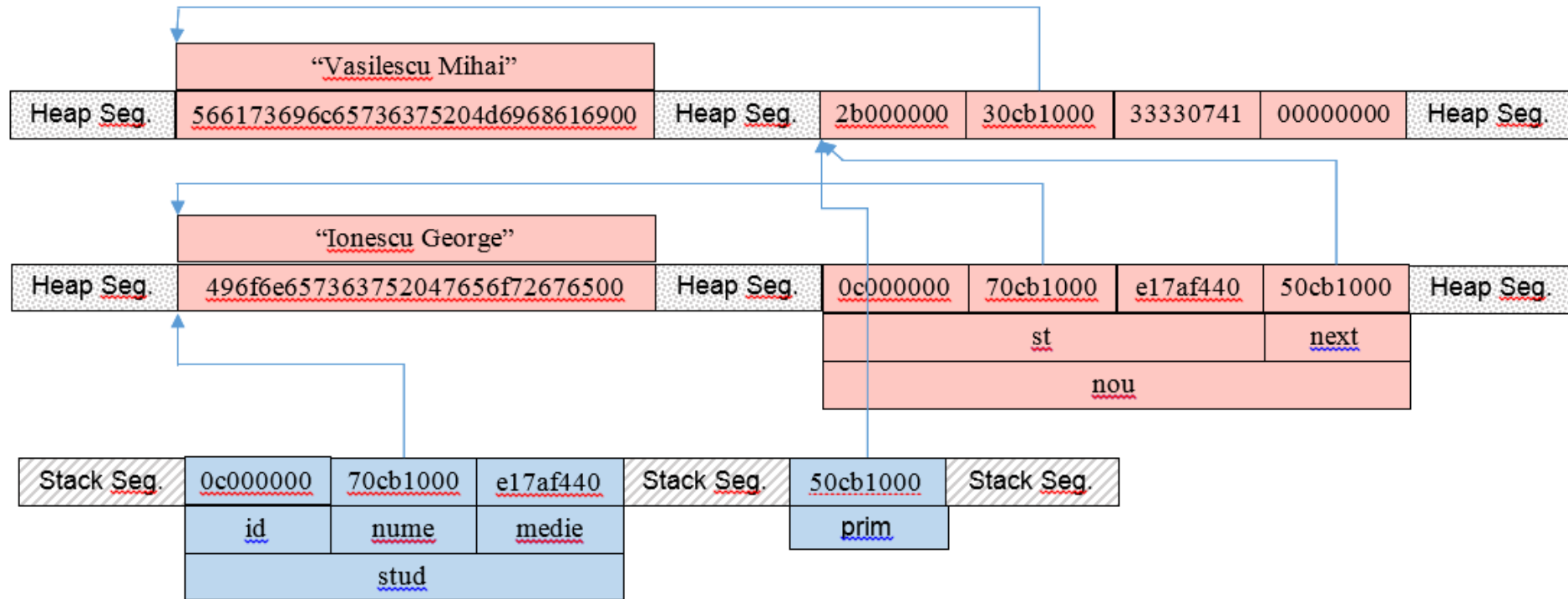
# LISTA SIMPLA

```
Nod* nou;
nou = (Nod*)malloc(1 * sizeof(Nod));
```



# LISTA SIMPLA

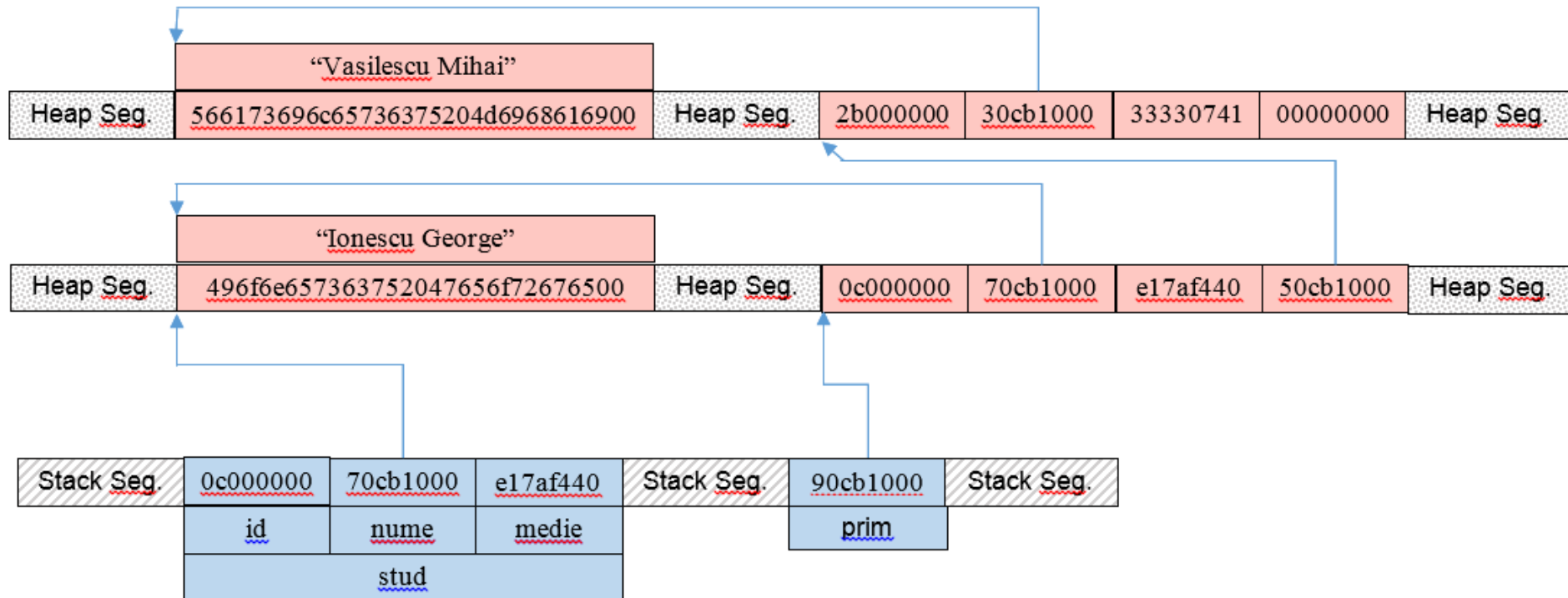
```
nou->st = stud;  
nou->next = prim;
```





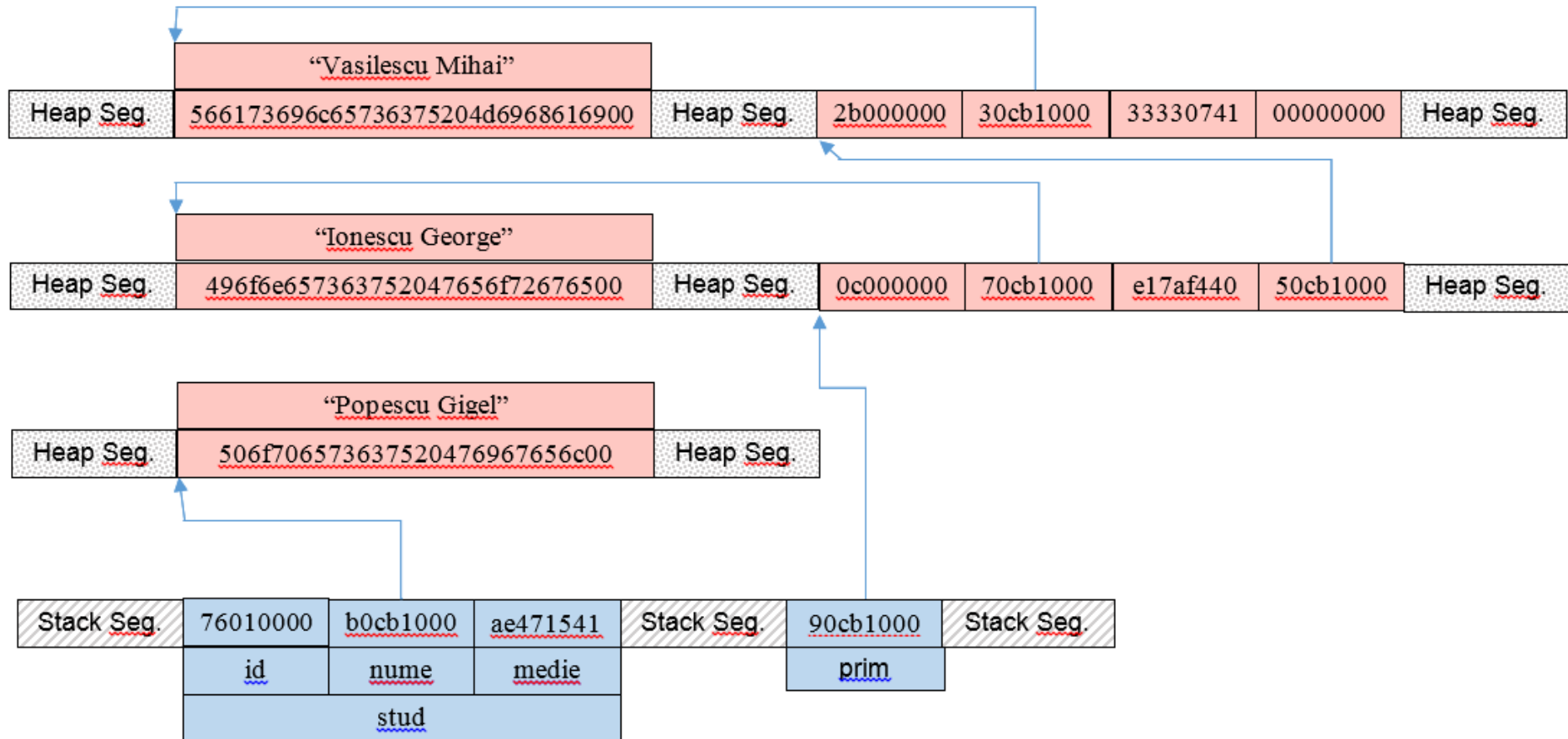
# LISTA SIMPLA

```
prim = inserareLista(prim, stud);
```



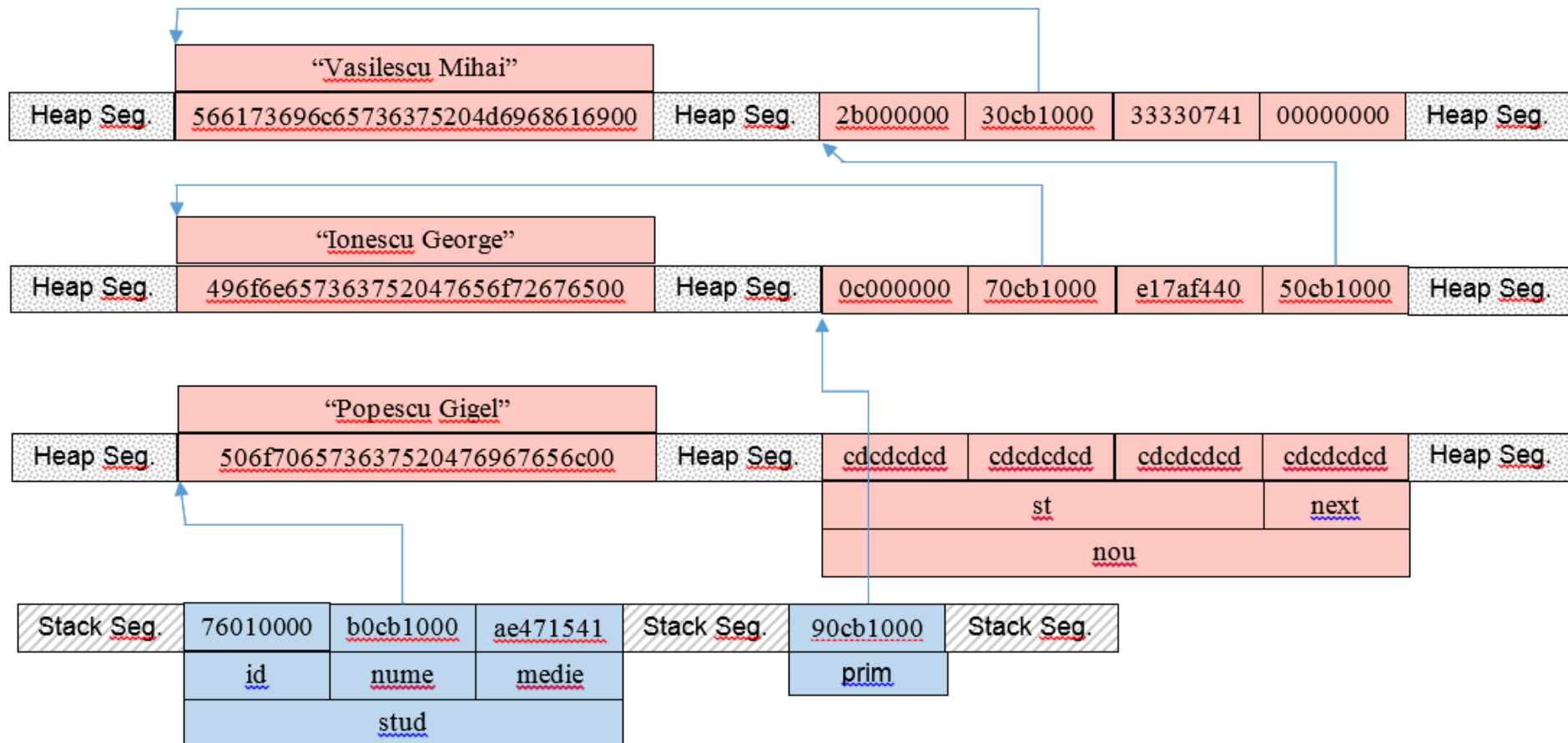
# LISTA SIMPLA

```
stud.id = atoi(token); // 374
stud.nume = (char*)malloc((strlen(token) + 1) * sizeof(char));
strcpy(stud.nume, token); // Popescu Gigel
stud.medie = atof(token); // 9.33
```



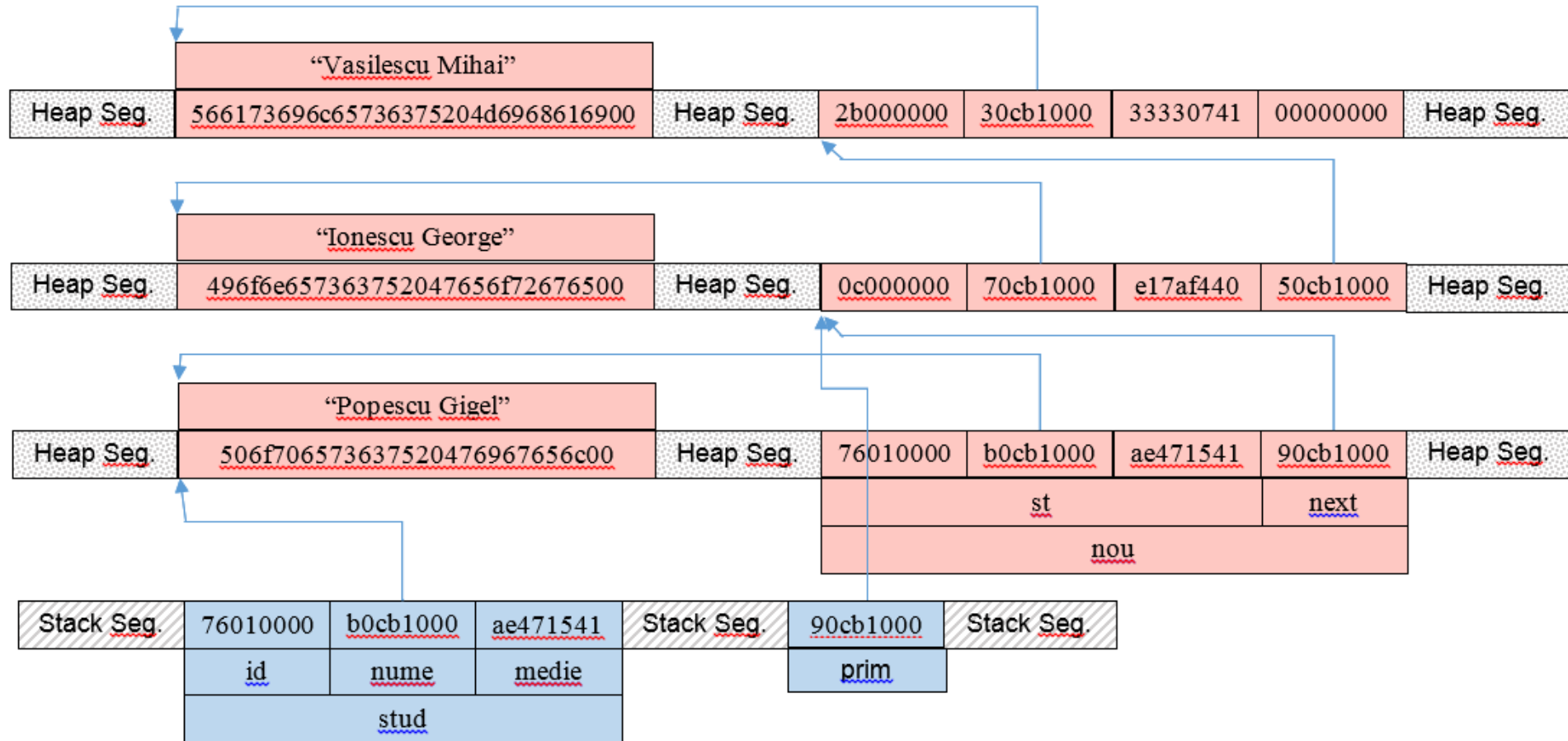
# LISTA SIMPLA

```
Nod* nou;  
nou = (Nod*)malloc(1 * sizeof(Nod));
```



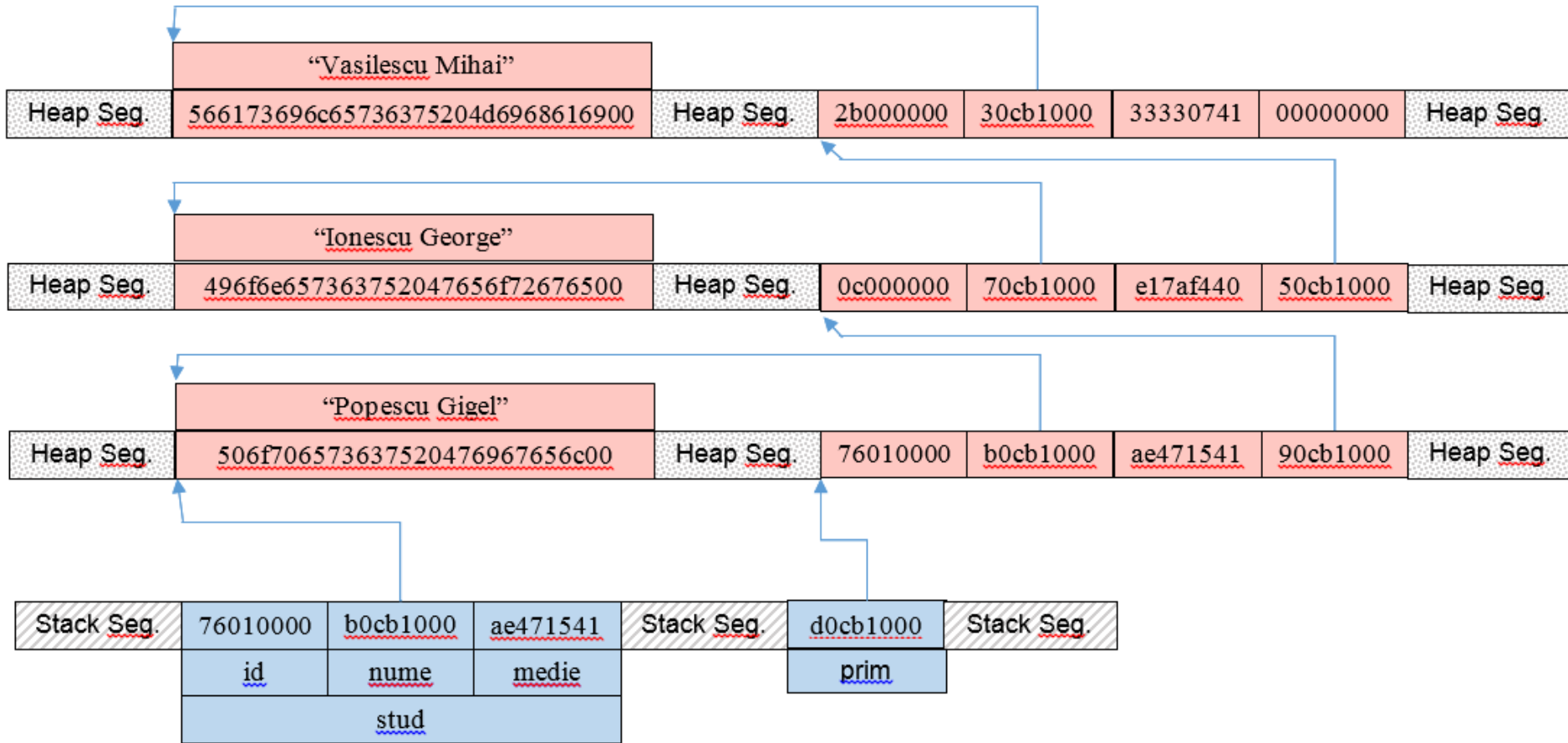
# LISTA SIMPLA

```
nou->st = s;  
nou->next = p;
```



# LISTA SIMPLA

```
prim = inserareLista(prim, stud);
```



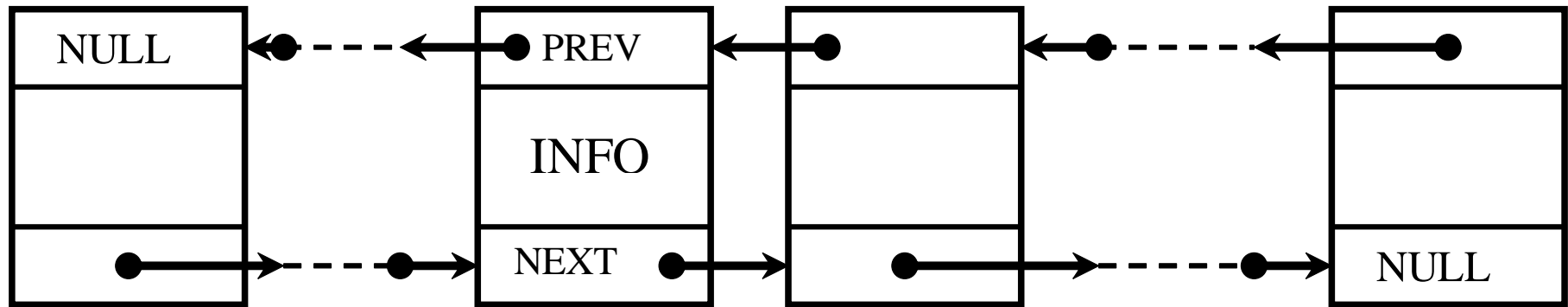
# LISTA DUBLA

- Structura liniara dinamica.
- Doua campuri cu informatii de legatura – nodurile succesori, respectiv predecesori ale campului curent
- Ultimul nod nu are succesori – informatia de legatura este nula (NULL sau 0).
- Primul nod nu are predecesori - informatia de legatura este nula (NULL sau 0).

# LISTA DUBLA

- Gestionata prin variabile pointer: una retine adresa primului nod din lista liniara, iar cealalta retine adresa ultimului nod din lista liniara.
- Posibilitatea gestionarii structurii printr-o structura articol ce incapsuleaza cei doi pointeri.

# LISTA DUBLA



Definirea structurii unui nod dintr-o lista dubla:

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



# LISTA DUBLA

Incapsularea adreselor primului si ultimului nod ale unei liste cu 0 noduri (lista empty):

```
struct ListaD {  
    NodD *prim, *ultim;  
};
```

Declararea unei liste duble cu 0 noduri (lista empty):

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

Alocarea de memorie heap pentru un nod al liste duble:

```
struct NodD *Nou = (struct NodD*)malloc(sizeof(struct NodD));
```

# LISTA DUBLA

Dezalocarea de memorie heap pentru un nod al liste duble:

```
free (Nou) ;
```

Initializarea si referirea informatiei utile dintr-un nod al listei duble:

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

Initializarea si referirea informatiilor de legatura dintr-un nod al listei duble:

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

# LISTA DUBLA

Referirea datelor din primul nod al listei duble:

```
lst.prim->inf;  
lst.prim->next;  
lst.prim->prev;
```

Referirea datelor din ultimul nod al listei duble:

```
lst.ultim->inf;  
lst.ultim->next;  
lst.ultim->prev;
```

# LISTA DUBLA

Operatii cu liste duble – similare operatiilor cu liste simple