Strukture podataka

Povezane liste

Sadržaj

- Uvod i motivacija
- Struktura povezane liste
- Operacije na povezanoj listi
  - Kreiranje povezane liste
  - Dodavanje elementa u listu
  - Obilazak liste
  - Uništavanje liste (dealokaija)
- Prosljeđivanje povezane liste kao argumenta funkcije
- Zaključak. Prednosti i nedostaci povezanih listi
- Literatura

Uvod i motivacija

- Lista
  - struktura podataka u kojoj se pohranjuje kolekcija podataka (niz podataka).
- Implementacija liste u jeziku C++:
  - polje
  - povezana lista

Uvod i motivacija

- Impelementacija liste pomoću polja – primjeri
  - Lista (polje) od 10 ocjena sudenta

```
int studentOcjene[10];
```

  - Lista (polje) temperatura u posljednja dva tjedna

```
double temperatura[14];
```

  - Lista (polje) od 100 sudenta (student je struktura)

```
student s[100];
```

  - Lista (polje) od MAX (konstanta) zaposlenika

```
zaposlenik z[MAX];
```

Uvod i motivacija

- Lista uz uporabu statičkog polja:

```
int mojePolje[10];
```

  - Moramo unaprijed odrediti veličinu polja (liste).

- Dinamički alocirano polje:

```
int n, *mojePolje;
cin >> n;
mojePolje = new int[n];
```

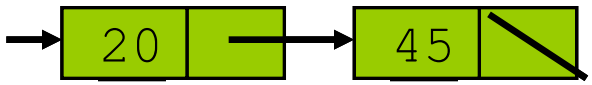
  - Alokacija polja (liste) bilo koje tražene veličine se vrši dinamički za vrijeme izvođenja programa.
  - Veličina polja nepromjenjiva nakon deklaracije

Uvod i motivacija

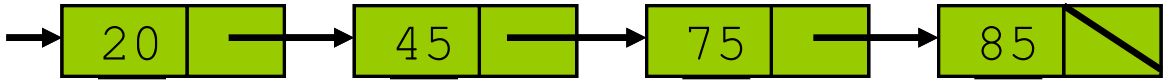
- Dinamički alocirano polje
  - Memorijski prostor zauzima se odjednom za sve elemente
  - Veličina polja ne mijenja se nakon što je polje dinamički alocirano
  - Podaci se pohranjuju na gomili
- Povezana lista
  - Memorijski prostor zauzima se za svaki element posebno
  - Veličina liste mijenja se dodavanjem i brisanjem elemenata, dinamička je
  - Lista može rasti i smanjivati, memorija se zauzima i oslobađa dinamički za točno onoliko elemenata koliko je u listi
  - Podaci se pohranjuju na gomili

Povezana lista: Osnovna ideja

- Povezana lista može rasti i smanjivati se - veličina je ograničena raspoloživim memorijskim prostorom (gomila).



dodaj(75), dodaj(85)

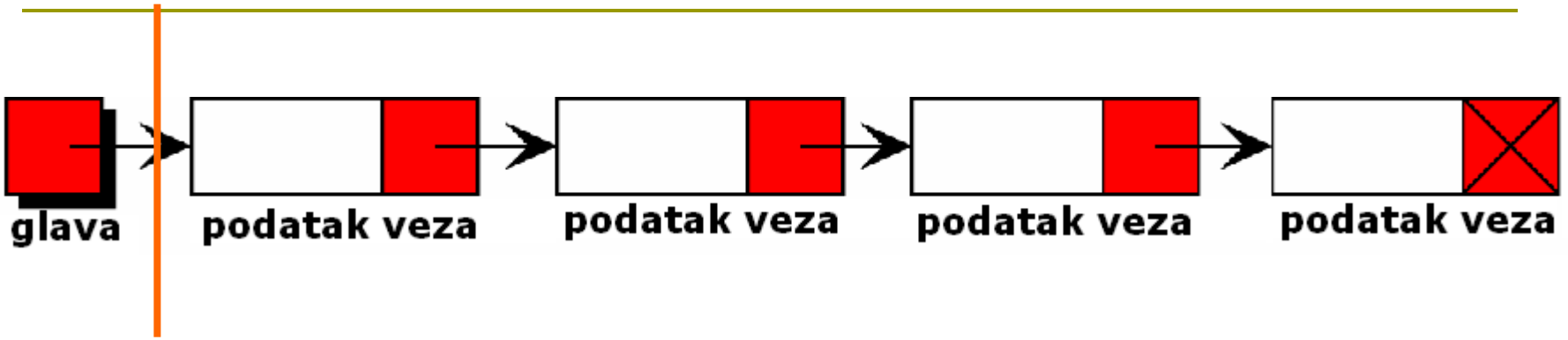


brisi(85), brisi(45), brisi(20)

Povezana lista: Osnovna ideja

- **Povezana lista** je kolekcija podataka, a svaki element liste sadrži **lokaciju slijedećeg elementa liste**.
- Elementi liste: čvorovi
- Svaki čvor sadrži dva dijela:
  - **Podatak** (pohranjeni podatak)
  - **Vezu** (adresa slijedećeg elementa liste)
- Izdvaja se pokazivač na prvi element liste
  - Uobičajeno se taj pokazivač naziva glava

Koncepti povezane liste



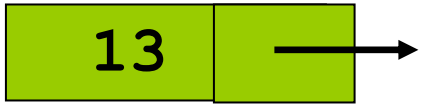
- Primjer: povezana lista sa 4 elementa od kojih se svaki sastoji od podatka i veze.
- **glava** je pokazivač na početak liste
- Križić kod zadnjeg elementa na mjestu veze označava NULL pokazivač (kraj liste).

Koncepti povezane liste

- Nema fizičke veze među elementima liste
  - nisu pohranjeni uzastopno u memoriji (kao elementi polja).
- Primjena pokazivača
  - Pokazivač na početak liste (glava)
  - Pokazivač za **obilazak** liste pri pretraživanju liste (pokazivač navigacije: **tekuci**, **temp**).
  - Ponekad se dodatno koristi pokazivač na kraj liste (rep)

Pitanja

- Kako biste u jeziku C++ implementirali povezanu listu?
- Kako biste definirali tip podatka koji prikazuje čvor povezane liste?



- Koji tip podatka biste koristili za povezivanje čvorova povezane liste?

Implementacija povezane liste

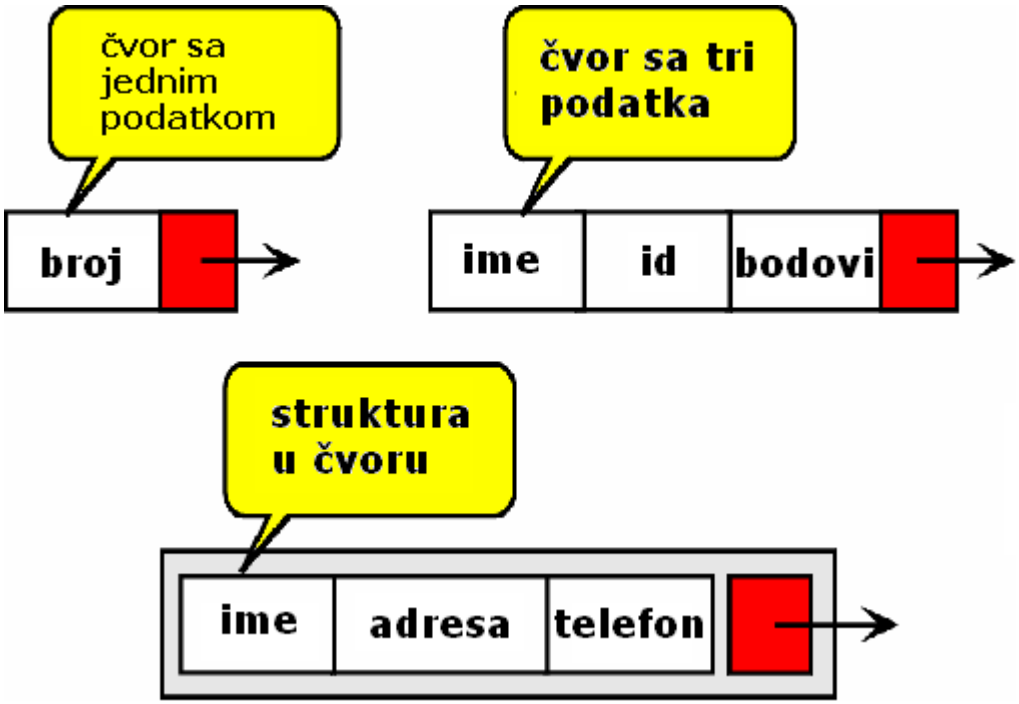
- Primjena pokazivača i dinamičke alokacije memorije
- Čvor je definira kao **struktura** koja sadrži podatkovnu komponentu i komponentu koja predstavlja vezu na idući element strukture
- Komponenta koja omogućava povezivanje između čvorova u listi implementira se pomoću pokazivača
- Prilikom dodavanja elementa u listu dinamički se alocira memorija samo za taj element
- Prilikom brisanja elementa iz liste dinamički se oslobađa memorija samo za taj element

Implementacija povezane liste

- Struktura čvor sadrži:
  - Podatak
    - Može biti bilo kojeg jednostavnog tipa
    - Može biti složeni tip podatka, struktura (npr. struktura student)
  - Vezu
    - Podatak koji pokazuje na idući element u listi
    - Pokazivač na čvor
      - Sadrži adresu idućeg čvora ili
      - Null-pokazivač ako je to zadnji čvor u listi

Implementacija povezane liste

- Tri različita primjera čvora:



Implementacija povezane liste

- Tri različita primjera čvora:
  - Čvor sa jednim podatkovnim članom (**broj**) i jednim članom koje sadrži vezu.
  - Čvor sa tri podatkovna člana (**ime, id, bodovi**) i jednim članom sa vezom.
  - Čvor sa jednim podatkovnim članom i članom sa vezom. Podatkovni član sadrži strukturu, koja se sastoji od više članova.

Implementacija povezane liste

- Struktura čvor

```
struct cvor {
   <tip_podatka> podatak;
   cvor* veza;
};
```

- Čvorovi koji čine povezanu listu su **rekurzivne strukture**:
  - struktura koja sadrži komponentu koja je pokazivač na primjerak iste te strukture

Implementacija povezane liste

- Struktura čvor
  - Primjer. Jednostavan čvor s podatkom cjelobrojnog tipa

```
struct cvor {
   //podatak cjelobrojnog tipa
   int podatak;
   //veza na idući element
   //ili null-pokzivač ako je zadnji
   cvor* veza;
};
```

Implementacija povezane liste

- Definicija strukture cvor:

```
struct cvor {
  int podatak;
  cvor* veza
};
```

- Alokacija memorijskog prostora za strukturu cvor:

```
cvor *pok;
pok = new cvor; //alocira se mem. prostor
```

- Dealokacija memorijskog prostora za strukturu cvor:

```
delete pok;
```

Implementacija povezane liste

- Pristup komponentama strukture cvor:

```
(*pok).podatak;  // pristup podatku
(*pok).veza;           // pristup vezi
```

- Alternativno označavanje:

```
pok -> podatak;  // pristup podatku
pok -> veza;           // pristup vezi
```

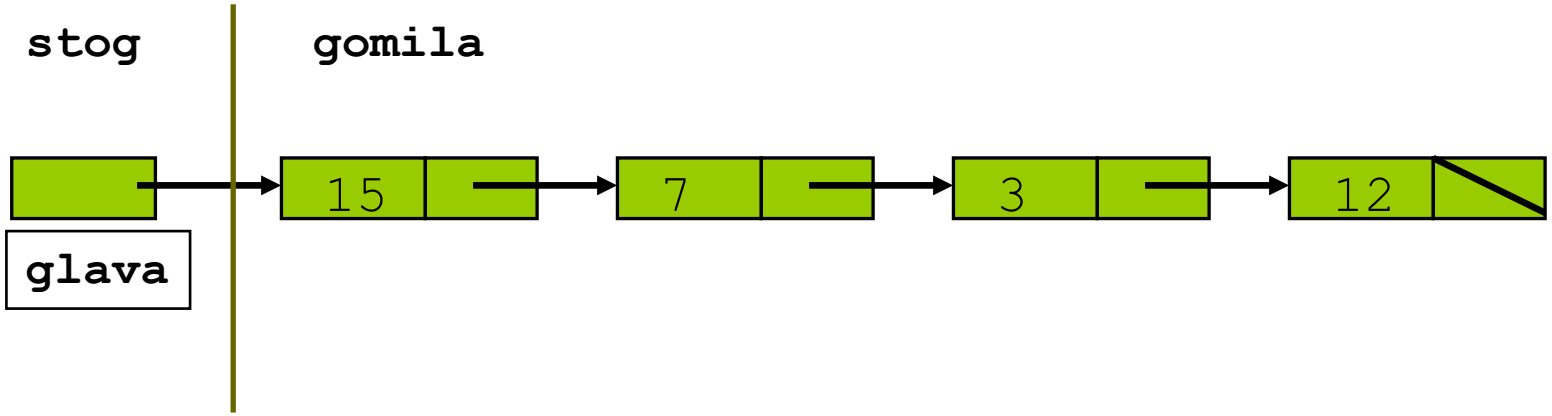
Implementacija povezane liste

- Definira se pokazivač koji pokazuje na početak liste (na prvi čvor):

```
cvor *glava;
```

- Kada je povezana lista prazna pokazivač **glava** ima vrijednost NULL (ili 0):

```
cvor *glava = NULL;
```

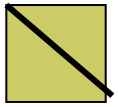


Implementacija liste

- Osnovne operacije s povezanom listom:
  - Kreiranje liste
  - Umetanje čvora
  - Obilazak liste
  - Brisanje čvora
  - Uništavanje liste
  - …

Kreiranje liste

- Opreracija se koristi za kreiranje prazne liste
- Deklarira se pokazivač na početak liste (glava)
  - Dok nije dodan niti jedan element u listu, pokazivač glava definira se kao null-pokazivač



glava

```
cvor *glava;
glava = 0;

//ili
//glava = NULL;
```

Umetanje čvora

- Operacija umeće čvor u povezanu listu.
- Za umetanje je potreban pokazivač na prethodnika čvora.
- Općenito, za zadanog prethodnika potrebni su sljedeći koraci umetanja:
  - Alokacija memorije za novi čvor
  - Povezivanje novog čvora s postojećom listom
    - Usmjeravanje novog čvora na njegovog nasljednika
    - Usmjeravanje prethodnog čvora na novi čvor

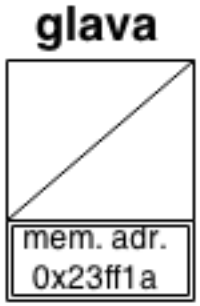
Umetanje čvora

- Postoje četiri različite situacije umetanja čvora u listu:
  1. Umetanje čvora u praznu listu
  2. Umetanje čvora na početak liste
  3. Umetanje čvora u sredinu liste
  4. Umetanje čvora na kraj liste

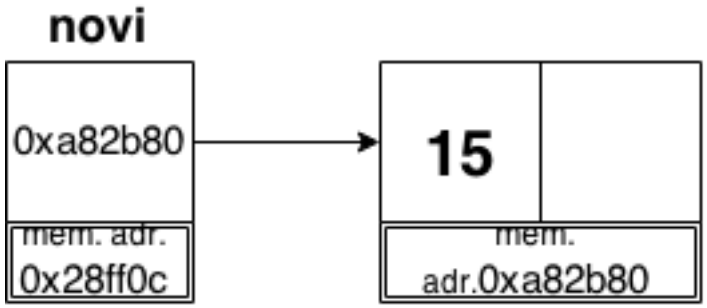
Dodavanje čvora u praznu listu

- Kada je pokazivač na glavu null-pokazivač, lista je prazna.
- Dodavanje čvora upraznu listu:
  - Alocira se memorijski prostor za novi čvor
    - Definira se podatak
    - Definira se veza novog čvora kao null-pokazivač
  - Pokazivač **`glava`** usmjeri se da pokazuje na novi čvor liste
- Rezultat
  - Nova lista ima samo jedan čvor (element)
  - Pokazivač **`glava`** pokazuje na taj čvor
  - Veza novog čvora je null-pokazivač jer je to kraj liste

Dodavanje čvora u praznu listu

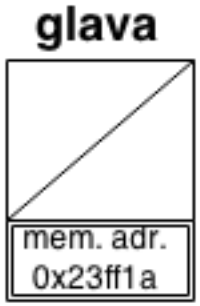


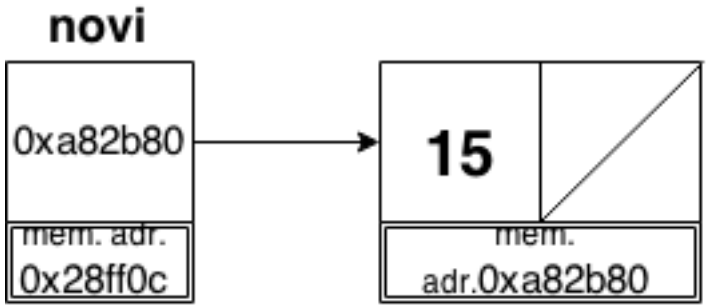

```
cvor * glava;
glava = 0;
```




```
cvor * novi;
novi = new cvor;
novi -> podatak = 15;
```

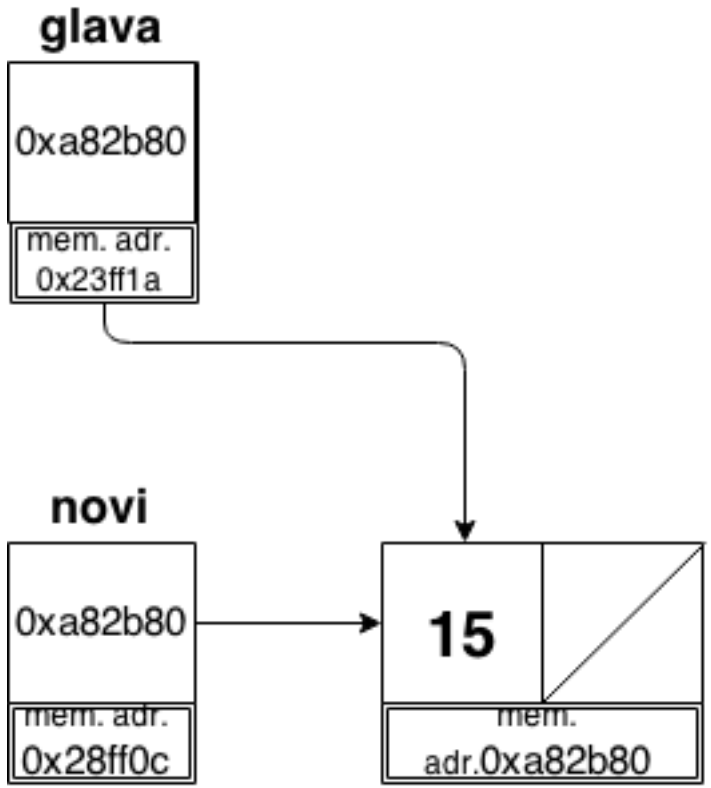
Dodavanje čvora u praznu listu






```
cvor * novi;
novi = new cvor;
novi -> podatak = 15;
novi -> veza = 0
```

Dodavanje čvora u praznu listu

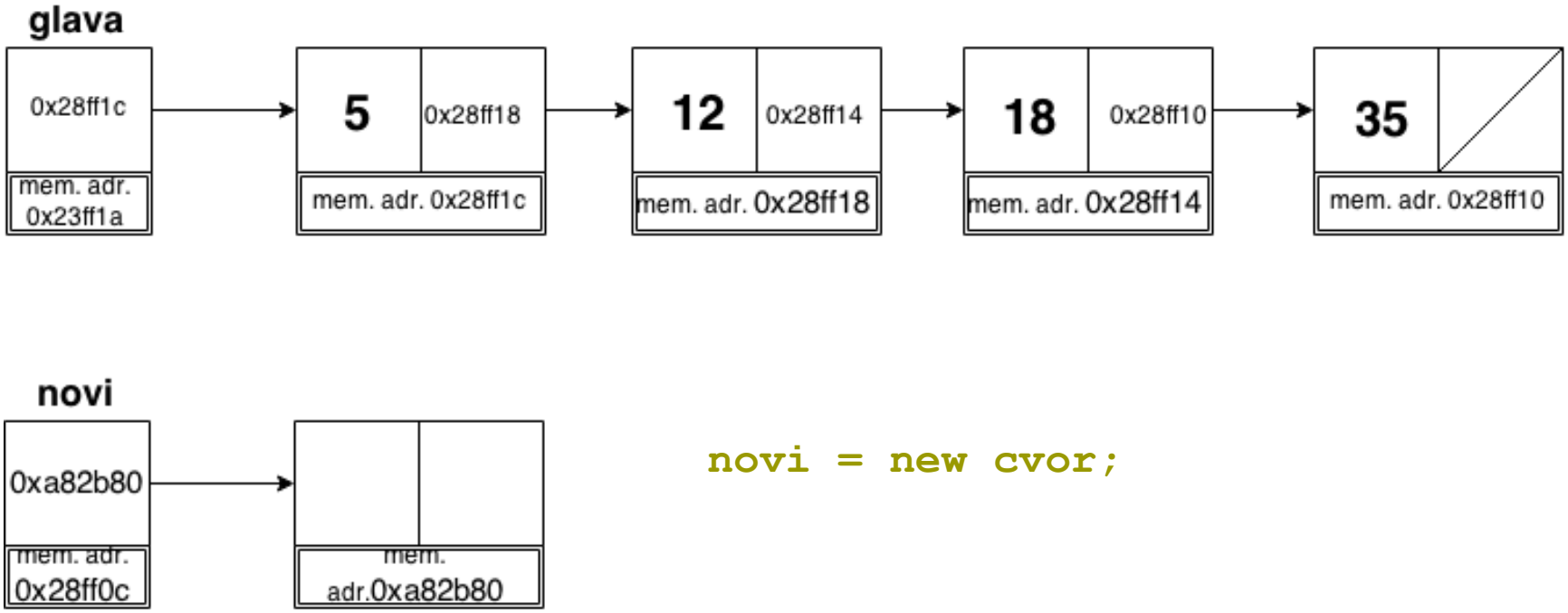



```
cvor * novi;
novi = new cvor;
novi -> podatak = 15;
novi -> veza = 0
glava = novi;
```

Dodavanje čvora na početak

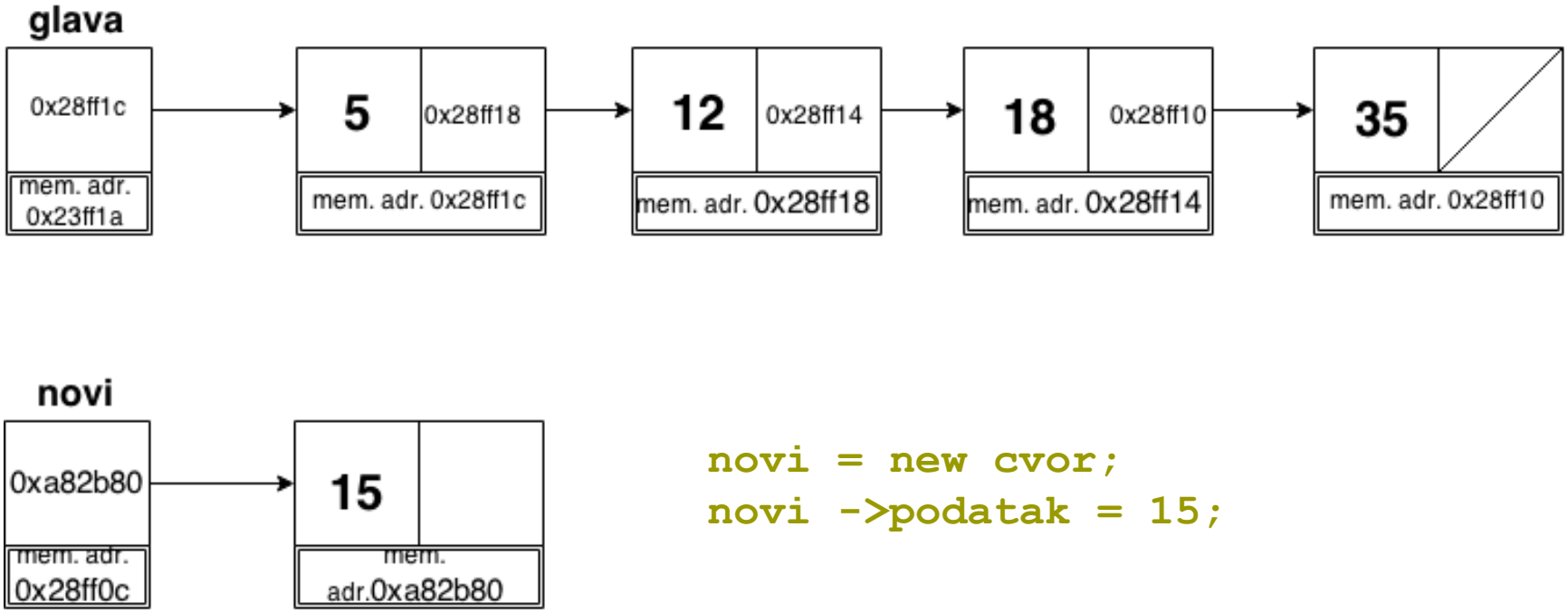
- Umećemo čvor na početak liste kada treba umetnuti novi čvor prije prvog elementa liste.
- Umetanje na početak liste:
  - Alocira se memorijski prostor za novi čvor
    - Definira se podatak
    - Postavi se veza novog čvora na pokazivač glave liste
  - Pokazivač **`glava`** usmjeri se da pokazuje na novi čvor liste
  - Rezultat
    - Ako je lista prazna, veza novog čvora postavlja se na NULL (null-pokazivač), a novi kreirani čvor je jedini element u listi.
    - Ako lista nije prazna, novi čvor mora pokazivati na ostatak liste.

Dodavanje čvora na početak – 1. korak



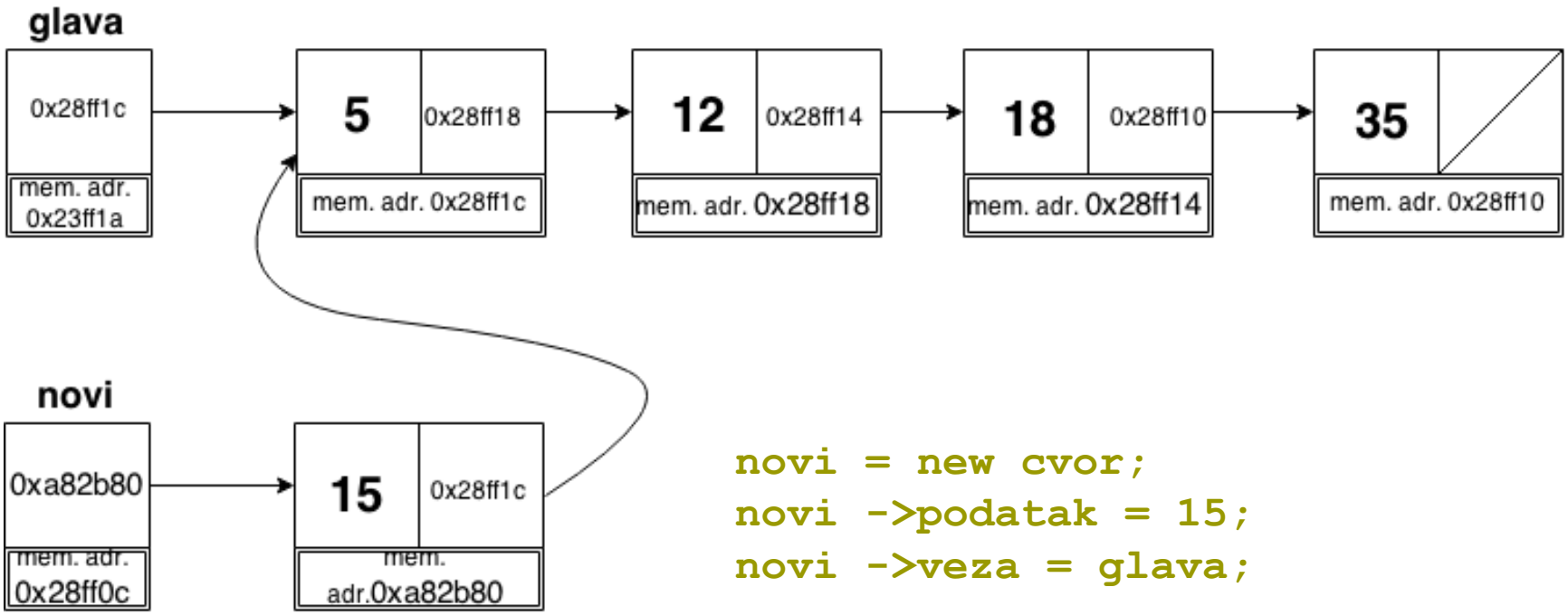

```
novi = new cvor;
```

Dodavanje čvora na početak – 2. korak



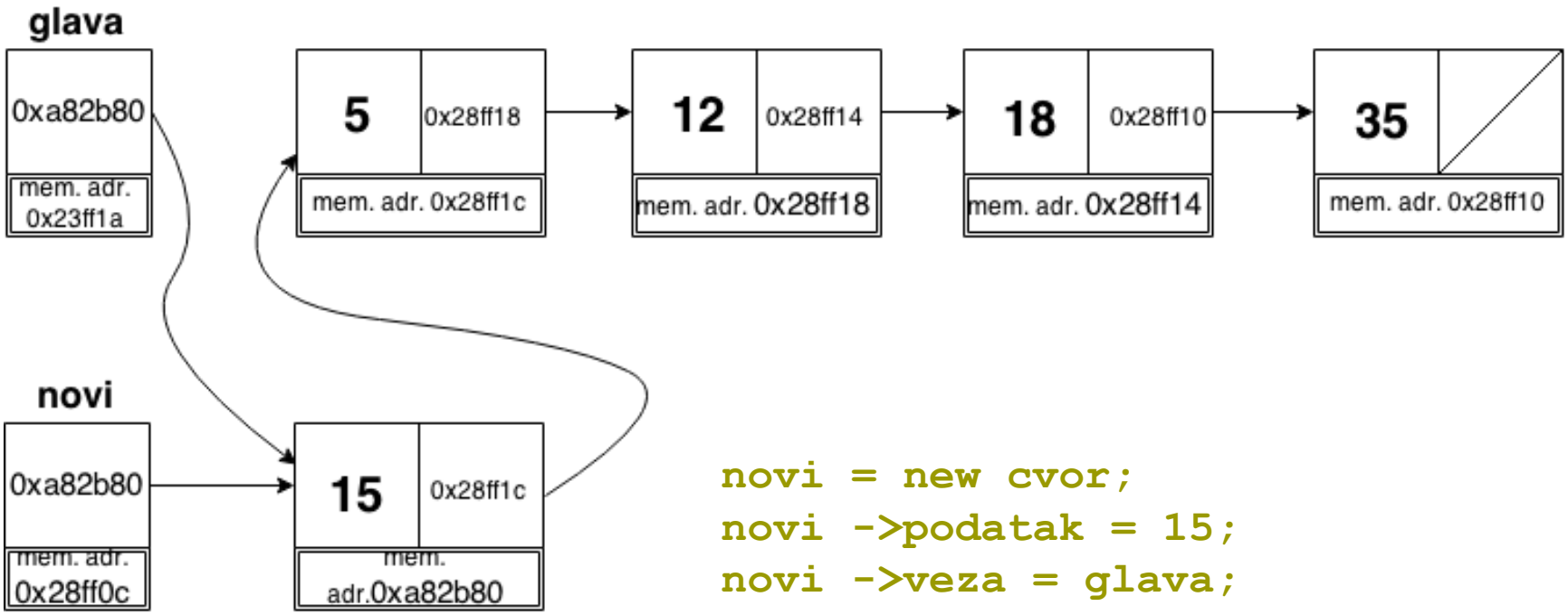

```
novi = new cvor;
novi ->podatak = 15;
```

Dodavanje čvora na početak – 3. korak



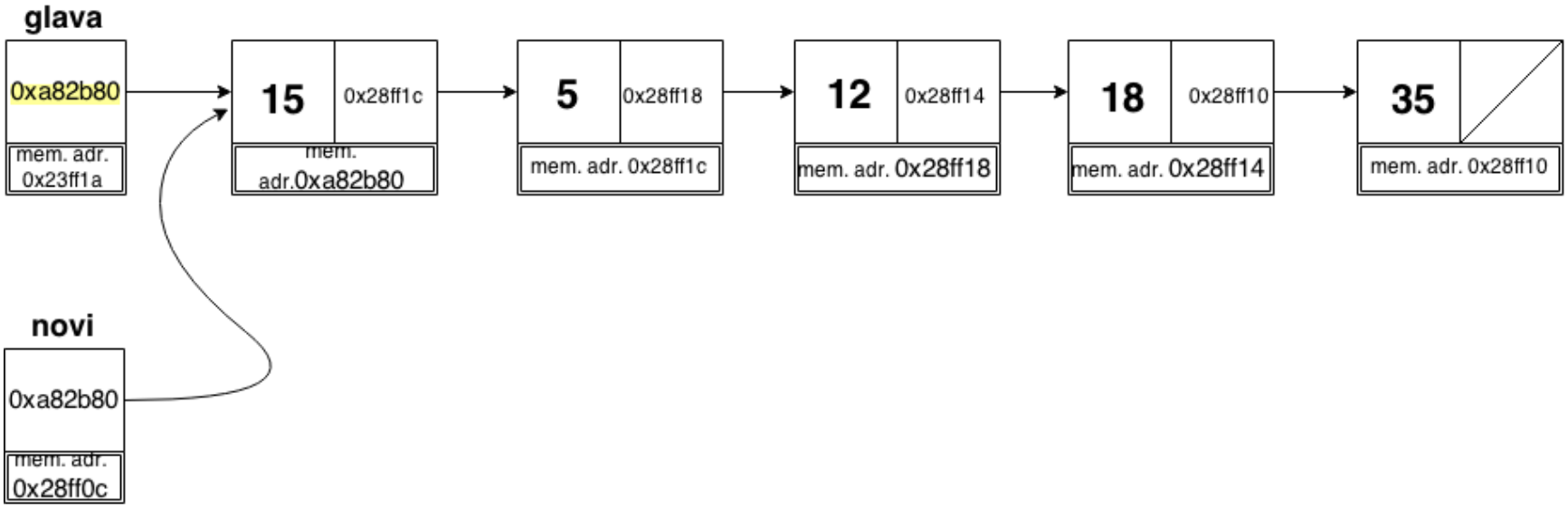

```
novi = new cvor;
novi ->podatak = 15;
novi ->veza = glava;
```

Dodavanje čvora na početak – 4. korak




```
novi = new cvor;
novi ->podatak = 15;
novi ->veza = glava;
glava = novi;
```

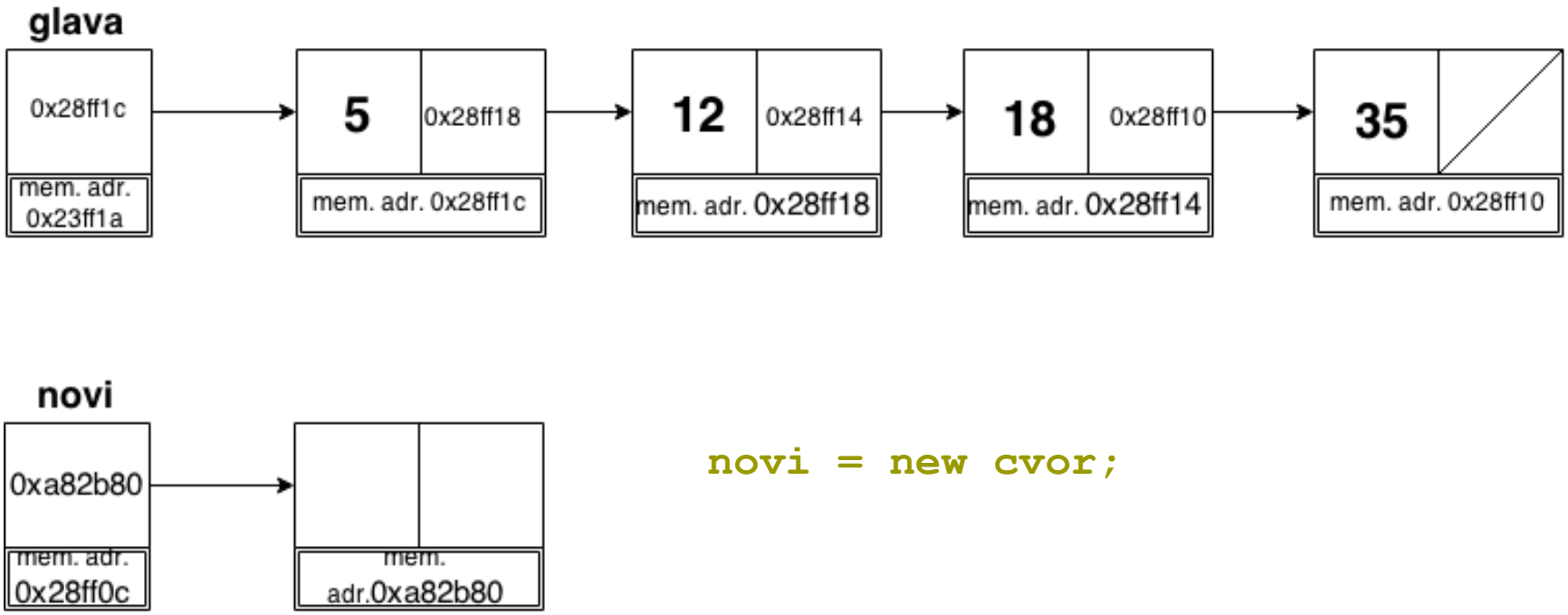
Dodavanje čvora na početak – rezultat



Dodavanje čvora u sredinu

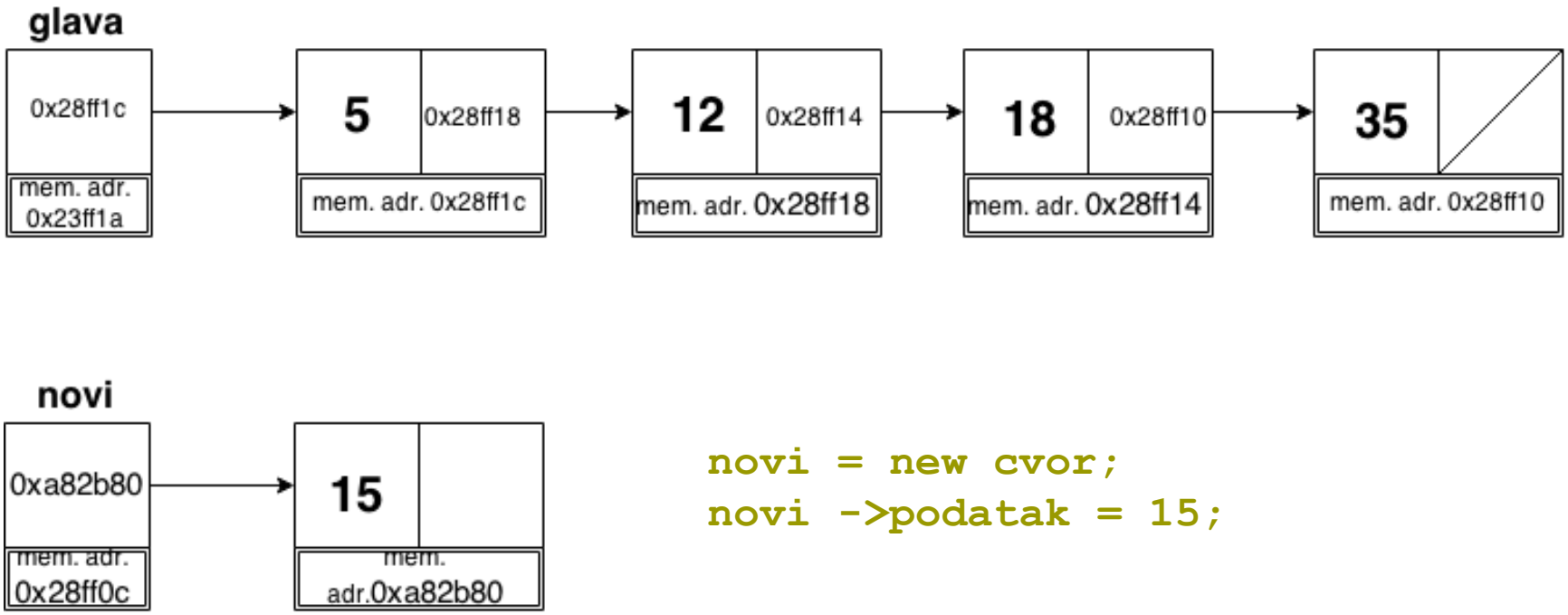
- Da bi umetnuli novi čvor između dva čvora, usmjerimo novi čvor na njegovog nasljednika i zatim usmjerimo prethodnika na novi čvor.
- Primjer
  - Želimo umetnuti novi čvor (17) nakon čvora (3) koji se nalazi u sredini liste
  - Na čvor nakon kojeg želimo umetnuti novi čvor potrebno je postaviti pokazivač prethodni (**`preth`**) radi povezivanja novog čvora s postojećom listom

Dodavanje čvora u sredinu – 1. korak



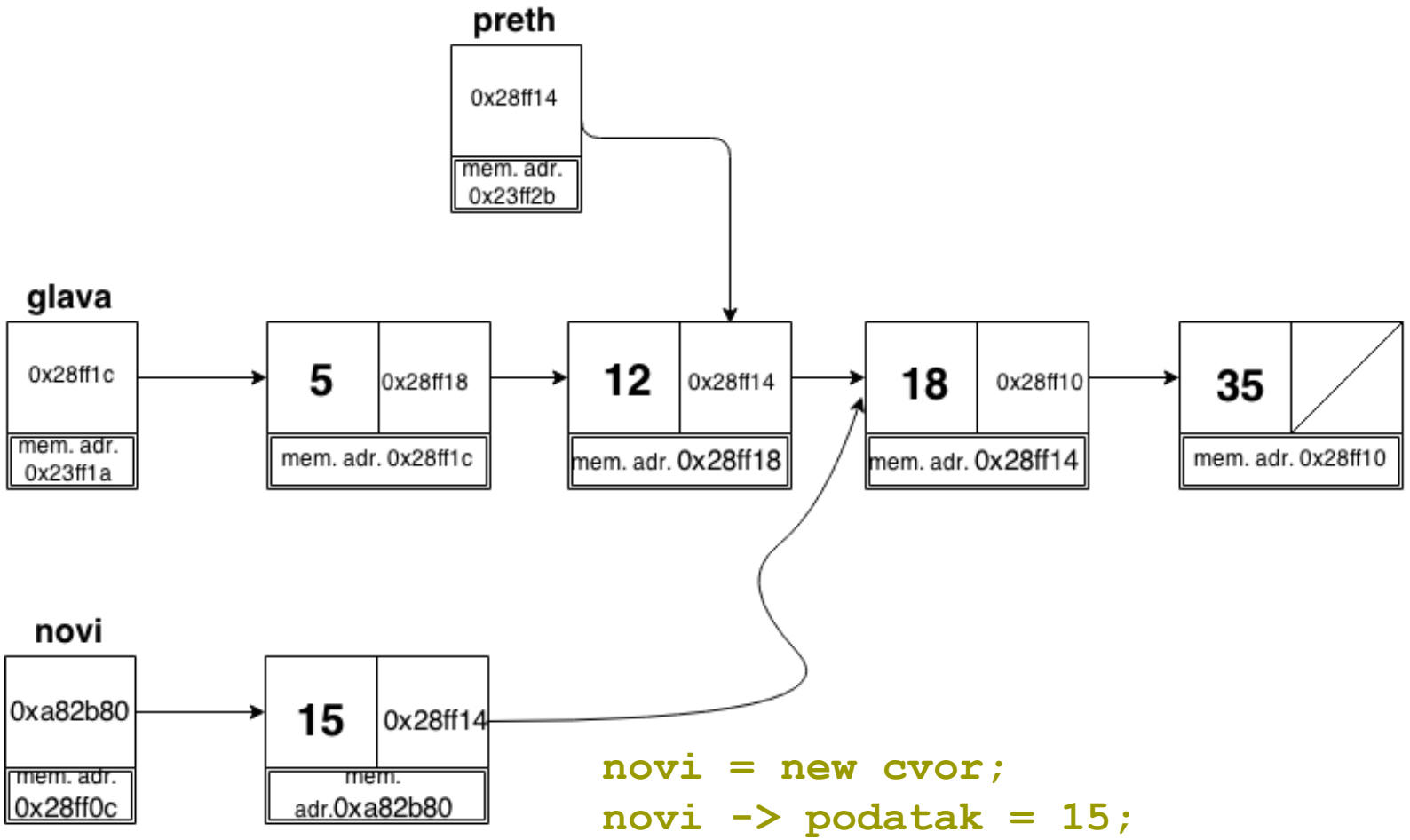

```
novi = new cvor;
```

Dodavanje čvora u sredinu – 2. korak



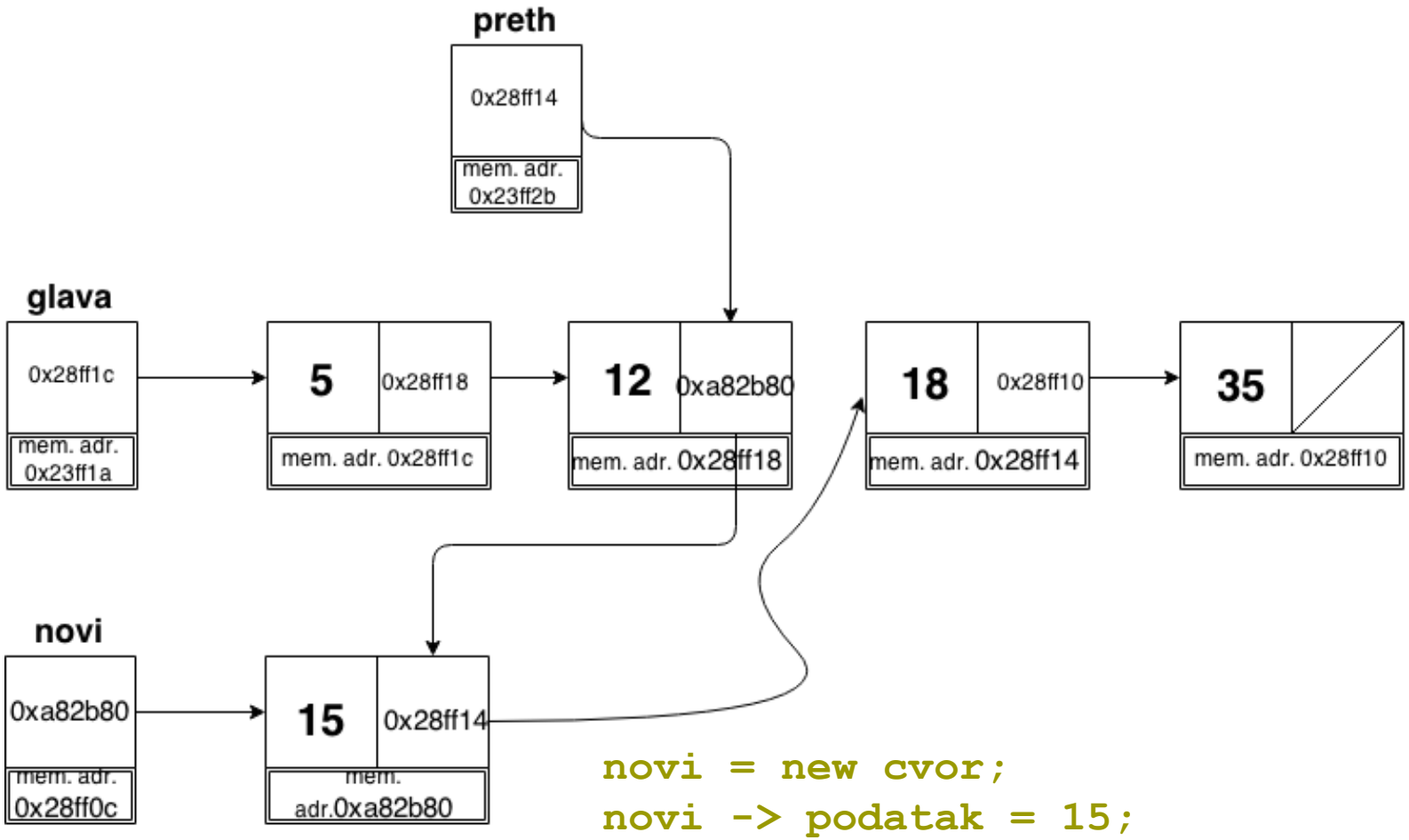

```
novi = new cvor;
novi ->podatak = 15;
```

Dodavanje čvora u sredinu – 3. korak



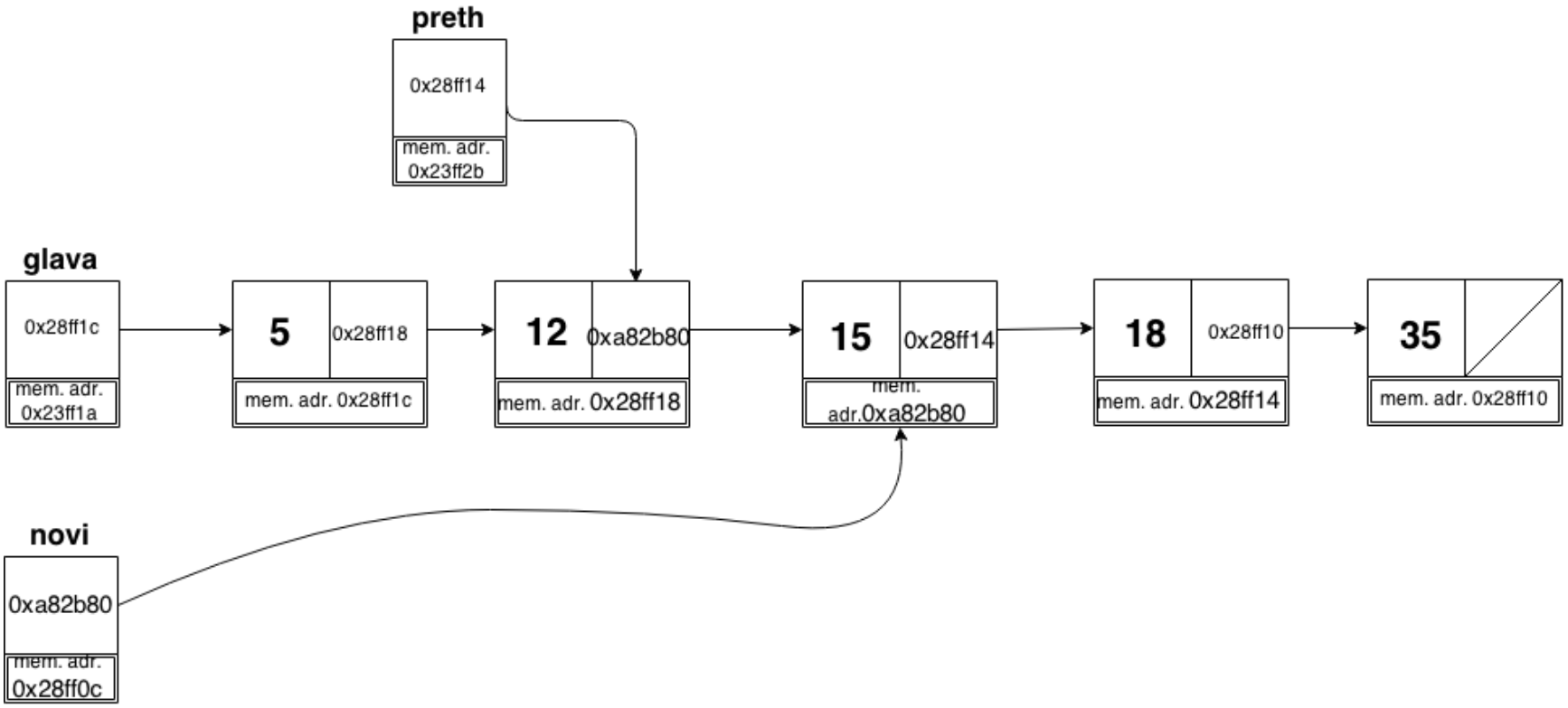

```
novi = new cvor;
novi -> podatak = 15;
Novi -> veza = preth -> veza;
```

Dodavanje čvora u sredinu – 4. korak




```
novi = new cvor;
novi -> podatak = 15;
novi -> veza = preth -> veza;
preth -> veza = novi;
```

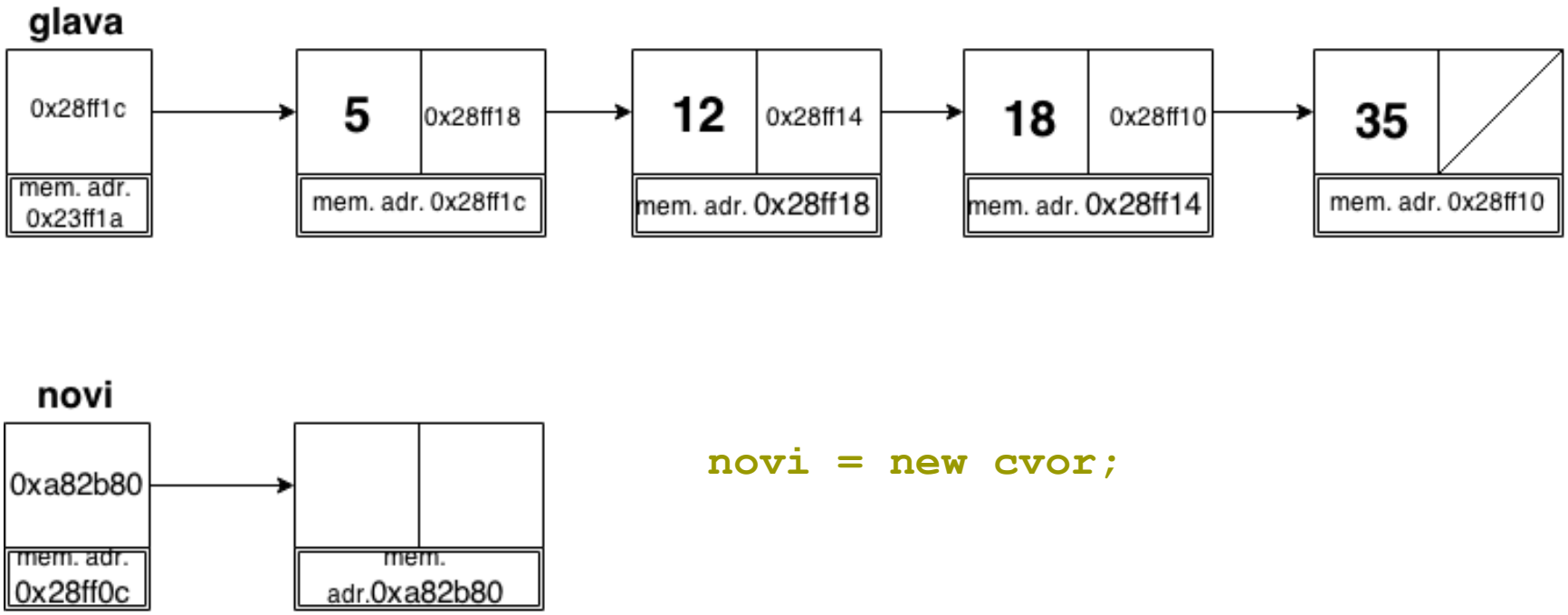
Dodavanje čvora u sredinu – rezultat



Dodavanje čvora na kraj

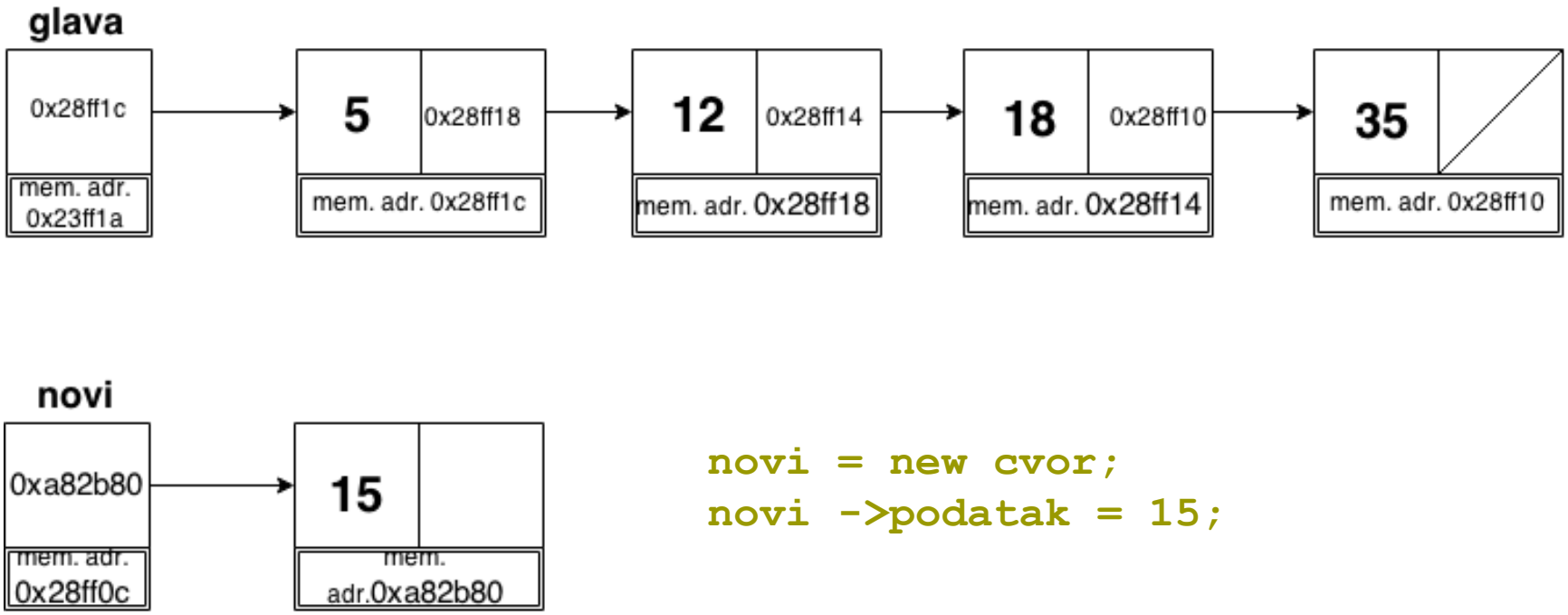
- Za umetanje čvora na kraj liste, moramo usmjeriti zadnji čvor liste na novi čvor i postaviti polje veze novog čvora na NULL.
- Uobičajeno je da se u takvim situacijama pamti i pokazivač na zadnji element liste koji se obično naziva **rep**.
- Ukoliko se kraj liste ne ‘pamti’ preko posebnog pokazivača rep, uvijek se može doći do kraja liste uz pomoć pomoćnog pokazivača koje pređe preko svih elemenata liste počevši od ‘glave’ do null-pokazivača

Dodavanje čvora na kraj – 1. korak



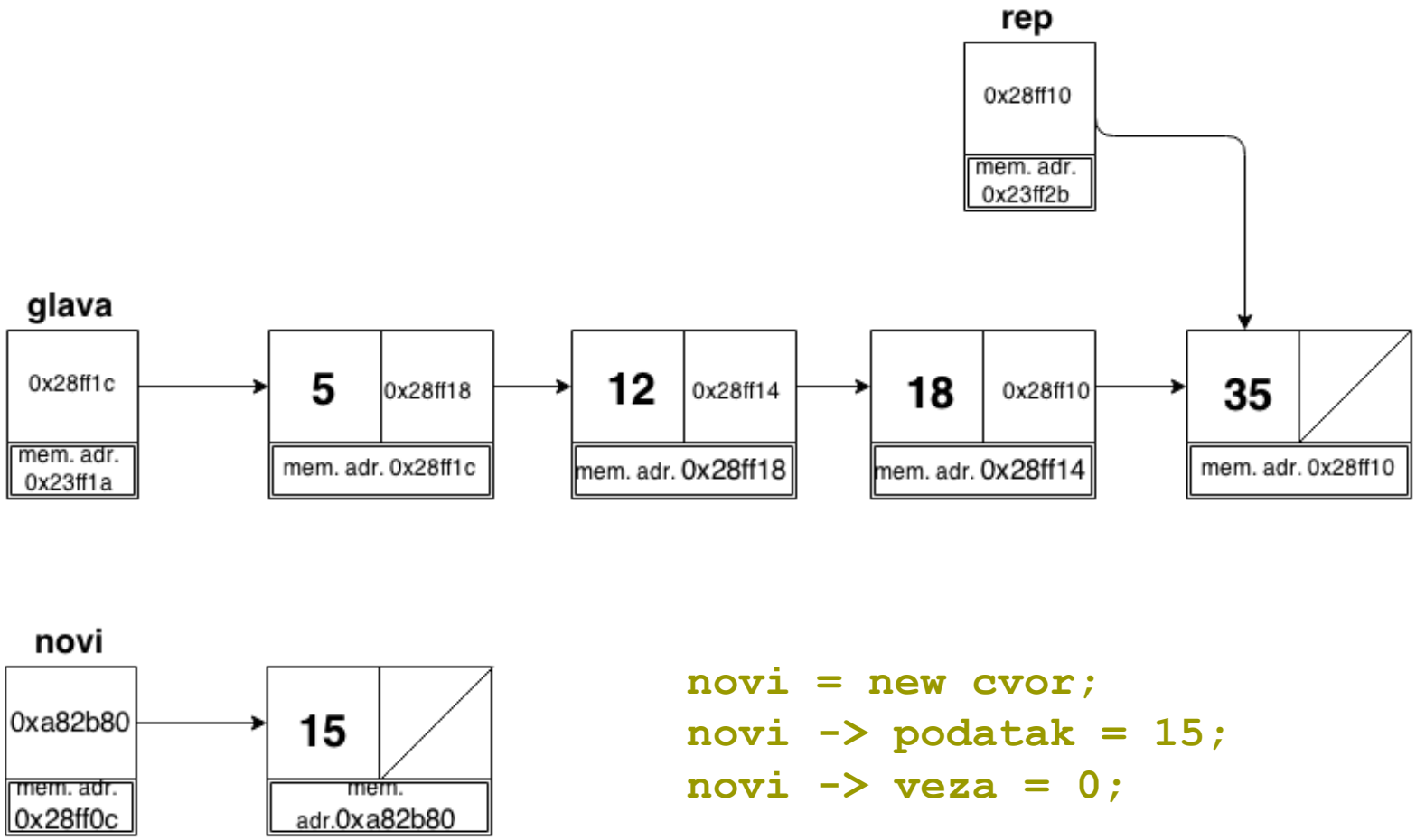

```
novi = new cvor;
```

Dodavanje čvora na kraj – 2. korak



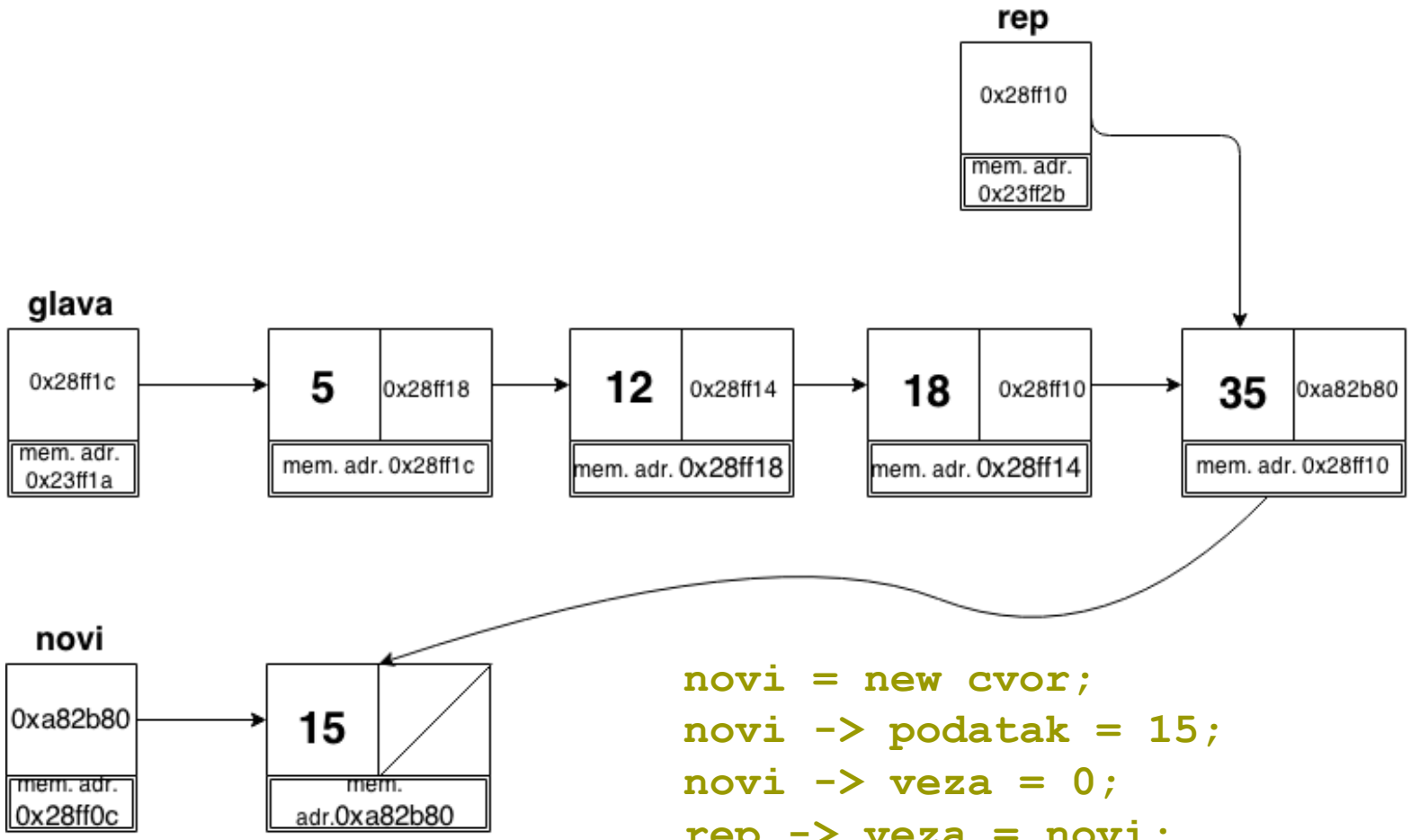

```
novi = new cvor;
novi ->podatak = 15;
```

Dodavanje čvora na kraj – 3. korak



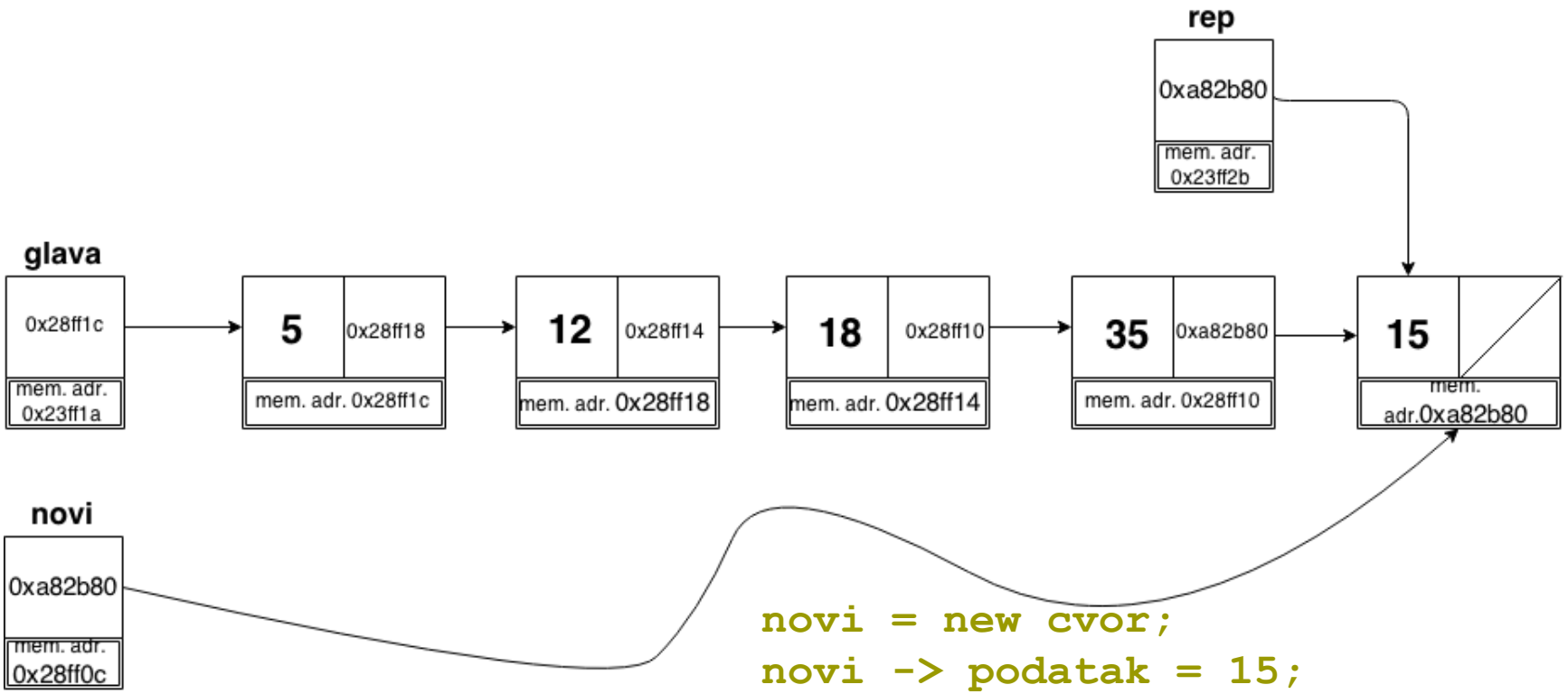

```
novi = new cvor;
novi -> podatak = 15;
novi -> veza = 0;
```

Dodavanje čvora na kraj – 4. korak




```
novi = new cvor;
novi -> podatak = 15;
novi -> veza = 0;
rep -> veza = novi;
```

Dodavanje čvora na kraj – rezultat




```
novi = new cvor;
novi -> podatak = 15;
novi -> veza = 0;
rep -> veza = novi;
rep = novi;
```

Algoritam umetanja čvora

- Uočimo da se četiri različita slučaja (umetanje u praznu listu, umetanje na početak, umetanje na kraj i umetanje u sredinu) reduciraju u kodu na dva slučaja:
  - Dodavanje na početak liste (prije prvog čvora) ili u praznu listu.
  - Dodavanje u sredinu ili na kraj liste.

Algoritam umetanja čvora

Psudokod:

```
if( preth null ) //preth je pokazivač na prethodni čvor
    //dodavanje u praznu listu ili na početak liste
    novi -> veza = glava
    glava = novi
else
    // dodavanje u sredinu ili na kraj
    novi -> veza = preth -> veza
    preth -> veza = novi
end if
```

Zadaci

- Implementirajte algoritme za umetanje elemenata u listu
  - Napišite funkciju koja omogućava korisniku unos i pohranjivanje elemenata (cijelih brojeva) u listu dok se ne učita broj 0. Svaki novi element liste neka se doda na početak liste.
  - Napišite funkciju koja omogućava korisniku unos i pohranjivanje elemenata (cijelih brojeva) u listu dok se ne učita broj 0. Svaki novi element liste neka se doda na kraj liste.

Obilazak liste

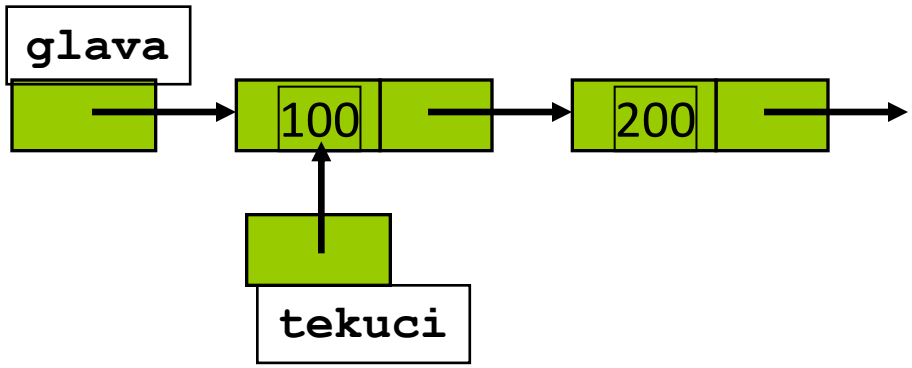
- Algoritmi za obilazak liste počinju s prvim čvorom i ispituju svaki čvor koji slijedi sve dok nije provjeren zadnji čvor.
- Obilazak liste potreban je u sljedećim situacijama:
  - promjena vrijednosti u čvoru,
  - ispis liste,
  - pretraživanje liste
  - zbrajanje elemenata liste,
  - računanje prosjeka elemenata liste
  - ...
- Svaki postupak koji zahtijeva da se cijela lista procesira koristi obilazak.

Obilazak liste

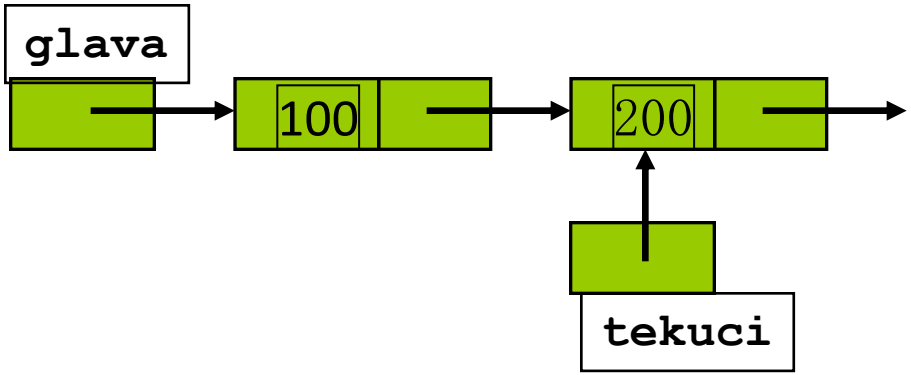
- Za obilazak liste trebamo **navigacijski pokazivač**.
- Ovaj pokazivač se koristi za pomak od čvora do čvora dok se obrađuju elementi.
- Uobičajeno je da se pokazivač za obilazak liste naziva **`tekuci`** (ili **`temp`**)
- Postavimo pokazivač obilaska na prvi čvor u listi (**`tekuci=glava;`**)
- Zatim obradimo prvi čvor i nastavimo s pomicanjem navigacijskog pokazivača i ispisom dok ne završimo postupak za sve čvorove.
- Kada je zadnji čvor obrađen, pokazivač obilaska postaje NULL i izvođenje petlje završava.

Obilazak i ispis povezane liste

```
tekuci = glava;
```




```
tekuci = tekuci->veza;
```



Obilazak i ispis povezane liste

- Npišite deklaraciju funkcije za obilazak (i ispis elemenata) povezane liste.
  - Koji su argumenti funkcije?

- Napišite definiciju funkcije za obilazak (i ispis elemenata) povezane liste.

Obilazak i ispis povezane liste

- Povezana lista se prikazuje obilaskom njezinih čvorova i prikazivanjem podatkovnih članova u čvorovima.

```
void PrikazListe(cvor * glava){
        cvor * tekuci;
        tekuci = glava;
        while(tekuci != 0){
                cout << tekuci->podatak << endl;
                tekuci = tekuci->veza;
        }
}
```

Prazna lista - provjera

- Kôd koji pišemo često pretpostavlja da lista nije prazna.
- Potrebno je provjeriti da li je lista neprazna prije početka pretraživanja, umetanja, brisanja i sl.
- Provjeravamo na slijedeći način:

```
if(glava != 0) // je li lista neprazna
// jednostavnije
if(glava)
```

- Obrnuto - je li lista prazna:

```
if(glava == 0) // je li lista prazna
// jednostavnije
if(!glava)
```

Oslobađanje memorije

- Kada nam lista više nije potrebna listu moramo uništiti
  - dealokacija povezane liste
  - postupak briše svaki čvor liste, oslobađa memoriju
  - potrebno je uvesti pomoćno pokazivač **`zaBrisanje`** kako bi se izbjeglo curenje memorije.
  - kôd:

```
cvor *zaBrisanje;
while (glava != 0){
   zaBrisanje = glava;
   glava = glava -> veza;
   delete zaBrisanje;
}
```

Prosljeđivanje povezane liste funkciji

- Koji argumeni se prosljeđuju funkciji koja omogućava dodavanje elementa na početak ili na kraj liste?
- Koji argumenti se prosljeđuju funkciji za uništavanje liste?

Prosljeđivanje povezane liste funkciji

- Kada se povezana lista prosljeđuje funkciji dovoljno je proslijediti pokazivač **glava** kao vrijednost. Preko pokazivača glava funkcija može pristupiti cijeloj listi.
- **Problem**: Ako se povezana lista u funkciji promijeni, npr. umetnemo ili izbrišemo prvi element liste, pokazivač glava više ne pokazuje na početak liste.
- **Rješenje**: Kada prosljeđujemo pokazivač glava, uvijek ga prosljeđujemo preko reference (ili se nova vrijednost pokazivača glava vraća primjenom anredbe return).

Prosljeđivanje povezane liste funkciji

- Što je pogrešno u sljedećim deklaracijama funkcija za dodavanje elementa u listu:

```
void dodajNaPoc(cvor *glava, int noviEl);
void dodajNaKraj(cvor *glava, int noviEl);
```

Prosljeđivanje povezane liste funkciji

```
void dodajNaKraj(cvor *& glava, int noviEl){

   cvor * novi = new cvor;
   novi -> podatak = noviEl;
   novi -> veza = 0;
   if(glava != 0){        // slučaj neprazne liste
       cvor * rep = glava;
       while(rep->veza != 0) //traži zadnji
             rep = rep->veza;
       rep->veza = novi;
   }
   else      // slučaj prazne liste
       glava = novi;
}
```

Pitanja

- Koje su prednosti povezanih listi?
  - Analizirajte prednosti u usporedbi s primjenom polja za pohranjivanje niza podataka
- Koji su nedostaci povezanih listi?
  - Analizirajte nedostatke u usporedbi s primjenom polja za pohranjivanje niza podataka

Povezane liste: prednosti

- 1. prednost: Dinamičnost:
  - Veličina statičkog polja ne može se mijenjati jer je veličina polja određena za vrijeme prevođenja.
  - Veličina dinamički alociranog polja ne može se mijenjati nakon što je alocirano.
  - Kako je povezana lista dinamička, njezina veličina se može povećati u svakom trenutku da bi se smjestio stvarni broj elemenata liste. Podaci su pohranjeni u dinamičku listu dinamički - svaki čvor je kreiran kada je potrebno
  - Povezana lista je prikladna kada je broj podataka koji se pohranjuju nepoznat.
  - Polje se može napuniti (zauzeti su svi elementi polja).
  - Povezana lista je puna samo kada ponestane memorije u koju se pohranjuju čvorovi.

Povezane liste: prednosti

- 2 prednost: Jednostavno umetanje i brisanje:
  - Iako su polja jednostavna za izvedbu i uporabu, ne omogućavaju učinkovito umetanje i brisanje sekvencijalnih podataka
  - Na primjer, zadatak: izbrišite broj 10 (ili neki proizvoljni element polja) u polju int polje[]={2,5,6,3,10,23,80} - nije jednostavno implementirati
  - Struktura povezane liste omogućava jednostavno umetanje i brisanje u listi

Povezane liste: nedostaci

- Povezane liste imaju određene nedostatke:
  - Općenito obilazak polja jednostavniji je od obilaska liste
  - Također, pristupanje i-tom elementu polja jednostavnije je od pristupanja i-tom elementu u listi
  - Moguće je izvoditi učinkovita pretraživanja polja (npr. binarno), ali to nije praktično sa povezanom listom.

Literatura

- Data Structures (A Pseudocode Approach with c), autori: R.h. Gilberg, B.A. Forouzan
- Nick Parlante: Linked List Basics
  - http://cslibrary.stanford.edu/103/LinkedListBasics.pdf
- Nick Parlante: Pointers and Memory
  - http://cslibrary.stanford.edu/102/PointersAndMemory.pdf
- Nick Parlante: Linked List Pronlems
  - http://cslibrary.stanford.edu/105/LinkedListProblems.pdf
- Šribar: str. 355. i 356. (bez koda, jer je obrađeno uz uporabu klasa)
- Vulin: str. 185. - 224. (modificirati sintaksu koda u C-u prema sintaksi C++-a: unos i ispis podataka, alokacija i dealokacija memorije)