Programiranje 2

Predavanje 08 - vezane liste, prvi dio

Matej Mihelčić

Prirodoslovno-matematički fakultet
Matematički odsjek

29. travnja 2023.

Strukture koje sadrže pokazivače i samoreferencirajuće strukture

- Pokazivač na objekt nekog tipa smije biti **član strukture**. To **omogućava povezivanje objekata različitih tipova**, ovisno o tipu pokazivača.
- Dozvoljeno je da pokazivač, član strukture, **pokazuje ne istu takvu strukturu** (sadrži pokazivač na element istog tipa).
- Struktura ne može rekurzivno sadržavati samu sebe.

Struktura koja sadrži jedan ili više članova koji su pokazivači na strukturu tog istog tipa zove se **samoreferencirajuća** struktura.

Samoreferencirajuće strukture su bitna posljedica korištenja pokazivača kao elementa strukture. Ovisno o broju i svrsi pokazivača možemo implementirati različite složene tipove podataka s rekurzivnom definicijom (različito od rekurzivno definirane strukture - nije moguće u C-u).
Primjeri samoreferencirajućih struktura su **vezane liste** i **binarna stabla**.

Samoreferencirajuće strukture - primjeri

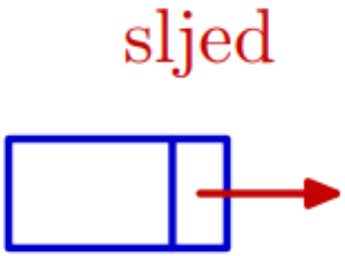
Svaki element samoreferencirajuće strukture je struktura koja ima dva bitna dijela:

- sadržaj - jedan ili više članova nekog tipa
- jedan ili više pokazivača na element istog tipa (strukturu)

Članovi strukture koji sadrže pokazivače na tu strukturu obično imaju standardna imena koja sugeriraju značenje pokazivača.

Element vezane liste

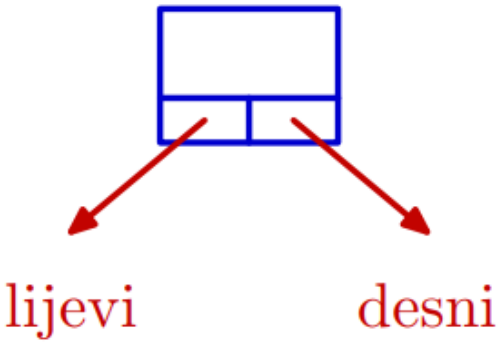
Element vezane liste, osim sadržaja, ima jedan pokazivač na element istog tipa (interpretiramo ga kao pokazivač na sljedeći element u listi). Često korištena imena za taj pokazivač (član strukture) su **sljed**, **veza**, **next**, **link**.



Ako zamislimo cijelu listu elemenata (kao na slici) koji se nalaze iza prikazanog, pokazivač možemo interpretirati i kao listu sljedbenika ovog elemenata (rekurzivna definicija).

Element binarnog stabla

Element binarnog stabla obično se zove čvor stabla. Osim sadržaja, ima dva pokazivača na element istog tipa (interpretiramo kao pokazivače na lijevo i desno dijete tog čvora). Standardna imena za te pokazivače su **lijevi**, **desni** ili **left**, **right**.



Pokazivač možemo interpretirati kao lijevo i desno podstablo prikazanog elementa (rekurzivna definicija).

Deklaracija elementa samoreferencirajuće strukture

- Koristimo `typedef` za deklaraciju tipa cijele strukture za element. Time izbjegnemo stalno pisanje riječi `struct` pri deklaraciji.
- Pretpostavimo da se sadržaj elementa sprema u jedan član strukture `info` tipa `sadrzaj`. Neka je tip `sadrzaj` ranije deklariran.
- Ovisno o vrsti strukture, u deklaraciju treba dodati jedan ili više pokazivača na takav element.

Deklaracija elementa vezane liste

Deklaracija tipa za element vezane liste takvih podataka:

```
1 typedef struct _element {
2         sadrzaj info; /* Sadrzaj. */
3         struct _element *sljed; /* Pokazivac. */
4 } element;
```

Član `sljed` je pokazivač na `struct _element`, a ne struktura, stoga definicija strukture **nije rekurzivna**. Bez znaka * bi imali pokušaj deklaracije strukture koja sadrži samu sebe (**nije dozvoljeno**).

Deklaracija elementa vezane liste — napomene

```
1 typedef struct _element {
2         sadrzaj info; /* Sadrzaj. */
3         struct _element *sljed; /* Pokazivac. */
4 } element;
```

U trenutku deklaracije pokazivača `sljed`, tip `struct _element` još nije potpuno određen. Međutim, memorija potrebna za spremanje pokazivača ne ovisi o tipu na koji pokazuje (stoga je definicija moguća). Nužno trebamo koristiti ime strukture `_element` za deklaraciju tipa pokazivača (tip `element` još nije definiran).

Primjer deklaracije elementa vezane liste

Deklariramo tip za vezanu listu koja treba sadržavati polje od 80 znakova.

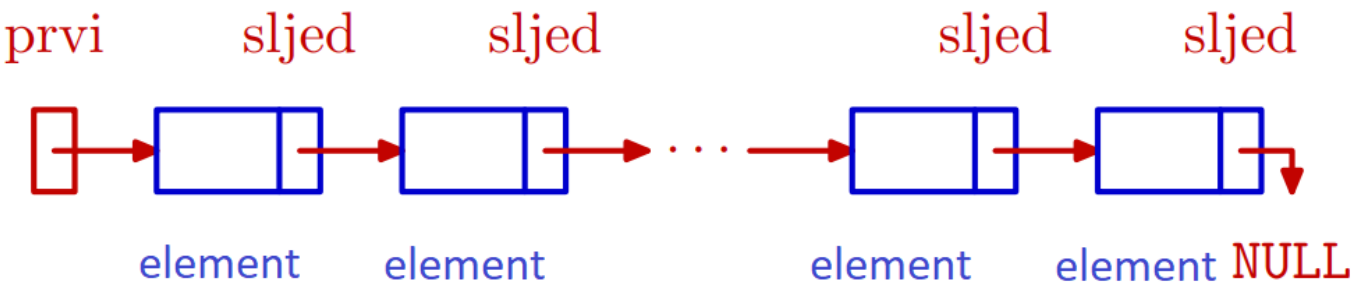
```
1 typedef struct _element {
2         char ime[80]; /* Sadrzaj. */
3         struct _element *sljed; /* Pokazivac. */
4 } element;
```

Ovdje koristimo ime `sljed` za pokazivač na sljedeći element liste.

Zadavanje vezane liste

Pošto vezana lista može biti i **prazna** (imati nula elemenata), ona se ne zadaje prvim elementom već pokazivačem na prvi element (ako postoji). Lista je prazna ako i samo ako je vrijednost tog pokazivača `NULL`.

Standardna imena za pokazivač na prvi element liste su `prvi`, `glava`, `first`, `head`.



Deklaracija pokazivača na element vezane liste

Pokazivač na **prvi** element liste možemo definirati kao:

```
1 struct _element *prvi; /* Pocetak liste. */
```

Gornju deklaraciju smijemo pisati i prije deklaracije tipa `element`.

Nakon deklaracije tipa `element` smijemo pisati:

```
1 element *prvi; /* Pocetak liste. */
```

Poželjno je uvesti i deklaraciju tipa za pokazivače na elemente liste.

```
1 typedef struct _element *lista;
```

Gornju deklaraciju smijemo pisati i **prije** deklaracije tipa `struct _element`.

Deklaracija pokazivača na element vezane liste

Skraćenu deklaraciju tipa ne smijemo napisati prije deklaracije tipa `element`:

```
1 typedef element *lista;
```

Na početku pišemo prvi oblik (sa struct), dalje koristimo tip lista (uključujući deklaraciju tipa).

Deklaracija vezane liste

Potrebne deklaracije tipova za vezanu listu su:

```
1 typedef struct _element *lista;
2
3 typedef struct _element {
4         sadrzaj info; /* Sadrzaj. */
5         lista sljed; /* Pokazivac. */
6 } element;
```

Listu deklariramo pokazivačem `prvi` kojega **inicijaliziramo** na `NULL` (prazna lista).

```
1 lista prvi = NULL; /* Pocetak liste. */
```

Vezana lista kao dinamička struktura

Nakon deklaracija, možemo definirati nekoliko varijabli tipa `element` i povezati ih u vezanu listu.

```
1 element a, b, c; /* Elementi liste. */
2 ...
3 prvi = &a;
4 strcpy(a.ime, "prvi"); /* NE: a.ime = "prvi"; */
5 a.sljed = &b;
6 strcpy(b.ime, "drugi");
7 b.sljed = &c;
8 strcpy(c.ime, "treci");
9 c.sljed = NULL;
```

U praksi se uglavnom kreiranje vezane liste radi drugačije, učitavanjem ili računanjem elemenata, spremanjem u dinamički alocirani objekt te povezivanjem tih objekata u listu.

Vezana lista kao dinamička struktura

Vezane liste i slične strukture su idealne za:

- **dinamičku** promjenu veza među elementima
- **dodavanje novih** i **izbacivanje postojećih** elemenata

Elementi takvih struktura se uvijek kreiraju **dinamičkom alokacijom** memorije.

```
prvi = (lista) malloc(sizeof(element));

if (prvi == NULL) {
     printf("Alokacija nije uspjela.\n");
     exit(EXIT_FAILURE); /* exit(1); */
}

strcpy(prvi->ime, "prvi");
prvi->sljed = NULL;
```

Deklaracija elementa binarnog stabla

Deklaracija **tipa** za element **binarnog stabla**:

```
1 typedef struct _cvor {
2         sadrzaj info; /* Sadrzaj. */
3         struct _cvor *lijevi; /* Pokazivac. */
4         struct _cvor *desni; /* Pokazivac. */
5 } cvor;
6 ...
7 cvor *korijen; /* Pokazivac na korijen. */
```

Početni element stabla obično se zove **korijen**. On nema roditelja u stablu, tj. nije dijete niti jednog elementa. U dinamičkoj strukturi, to je pokazivač na početni element stabla.

Deklaracije tipova za binarno stablo

```
1 typedef struct _treenode *Treeptr;
2
3 typedef struct _treenode {
4 ... /* Sadrzaj. */
5       Treeptr left; /* Pokazivac. */
6       Treeptr right; /* Pokazivac. */
7 } Treenode;
8 ...
9 Treeptr root; /* Pokazivac na korijen. */
```

Binarno stablo može biti prazno, u tom slučaju je `root = NULL`.

Operacije nad vezanim listama

Osnovne operacije nad vezanom listom su:

- kreiranje i uništavanje elemenata
- dodavanje novog elementa u listu
- brojanje elemenata liste
- ispis liste
- pretraživanje liste (prolaz kroz listu)
- izbacivanje ili brisanje elementa iz liste
- spajanje (konkatenacija) dvije liste
- sortiranje liste

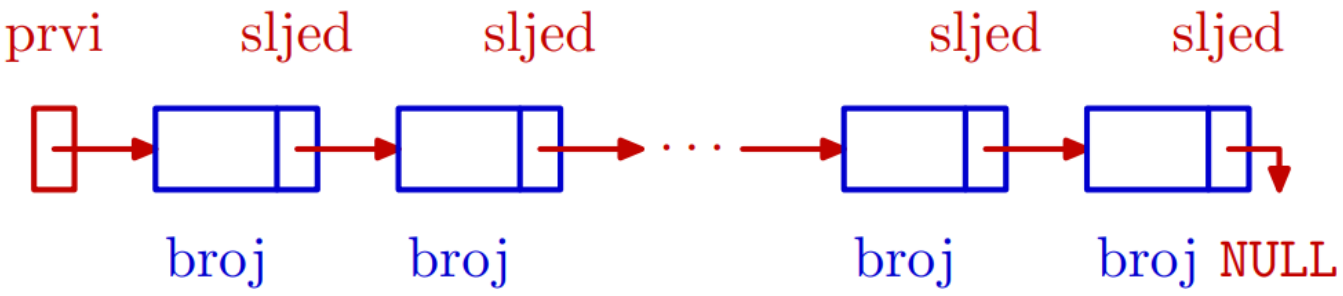
Navedene operacije ćemo implementirati u sljedećim primjerima.

Lista u primjerima

U primjerima ćemo koristiti vezanu listu **cijelih brojeva**.

```
1  /* Tip za pokazivac na element liste. */
2  typedef struct _element *lista;
3
4  /* Tip za element liste. */
5  typedef struct _element {
6       int broj; /* Sadrzaj je broj. */
7       lista sljed; /* Pokazivac na sljedeci */
8  } element; /* element u listi. */
9
10 lista prvi; /* Pokazivac na pocetak liste. */
```

Lista u primjerima

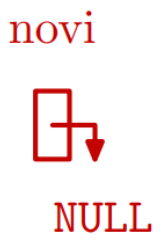


Operacije **dodavanja** i **izbacivanja** elemenata najlakše se rade na **početku** liste, zbog sekvencijalnog pristupa elementima.

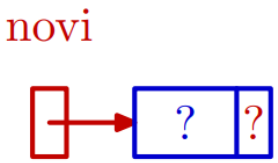
Pristup proizvoljnom elementu liste (kod nepraznih listi), moguć je preko pokazivača `prvi`. Potrebno je iterirati do elementa (pomoćnim pokazivačem `pom`), koristeći pokazivače `sljed`.

Sve **funkcije** koje rade s elementima vraćaju **pokazivač** na element (objekt tipa `lista`).

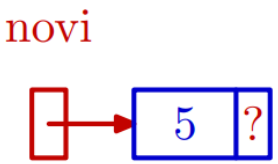
Kreiraj element



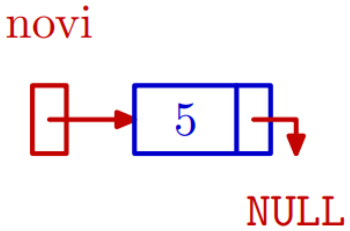

```
lista novi = NULL;
```




```
novi = (lista) malloc(sizeof(element));
```




```
novi->broj = broj;
```




```
novi->sljed = NULL;
```

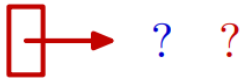
Funkcija kreiraj_novi

```
lista kreiraj_novi(int broj){

  lista novi = NULL;
  novi = (lista) malloc(sizeof(element));

  if (novi == NULL) {
       printf("Alokacija nije uspjela.\n");
       exit(EXIT_FAILURE); /* exit(1); */
  }

  novi->broj = broj;
  novi->sljed = NULL;
  return novi;
}
```

Obriši element

stari

stari




```
free(stari);
```

stari

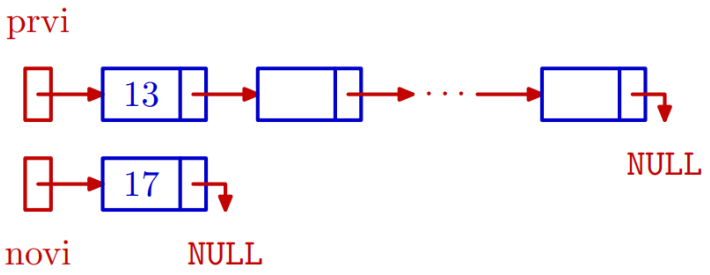
NULL

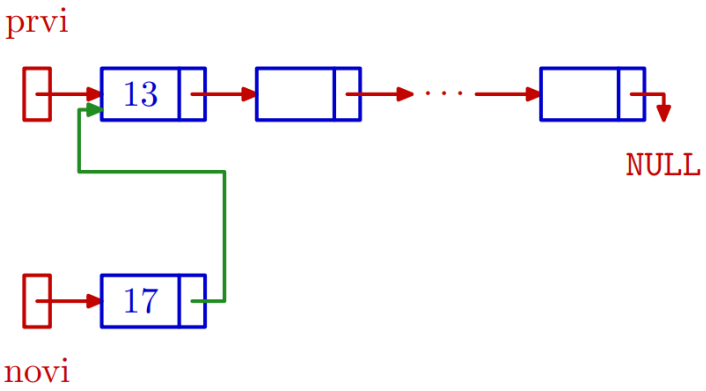
```
stari = NULL;
```

Funkcija obrisi_element

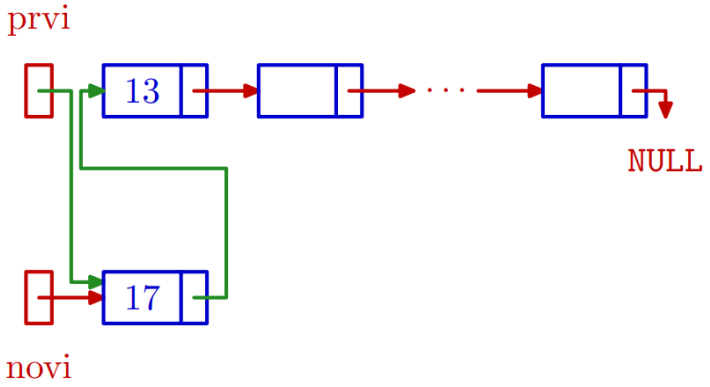
```
1 lista obrisi_element(lista stari){
2     free(stari);
3     return NULL; /* Umjesto stari = NULL; */
4 }
```

Ubaci na početak






```
novi->sljed = prvi;
```




```
prvi = novi;
```

Funkcija ubaci_na_pocetak

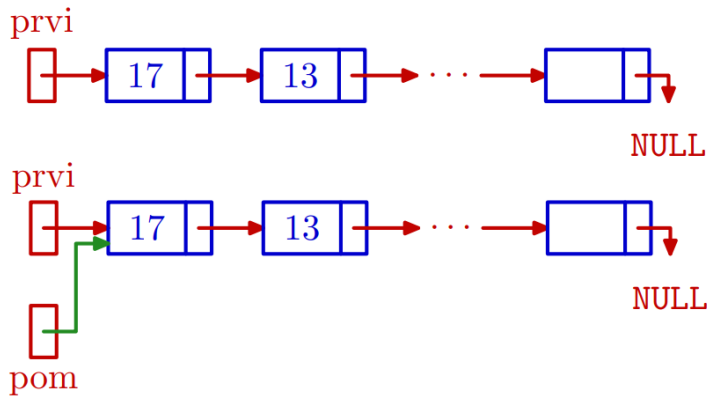
```
1 lista ubaci_na_pocetak(lista prvi, lista novi){
2 /* Ne provjerava novi != NULL. */
3     novi->sljed = prvi;
4     prvi = novi;
5     return prvi;
6 }
```

Ideja poziva svih funkcija za rad s listom:

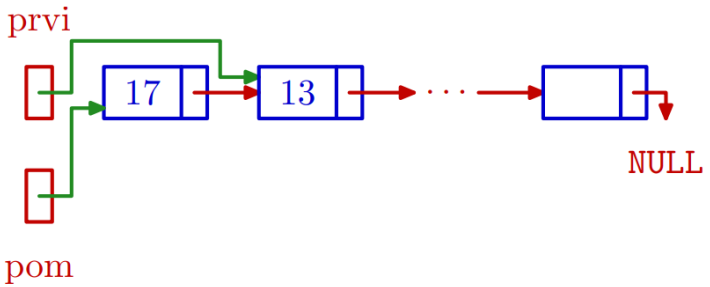
`prvi = funkcija_na_listi(prvi, ...);`

Dozvoljeno je da je lista **prazna** na ulazu i izlazu.

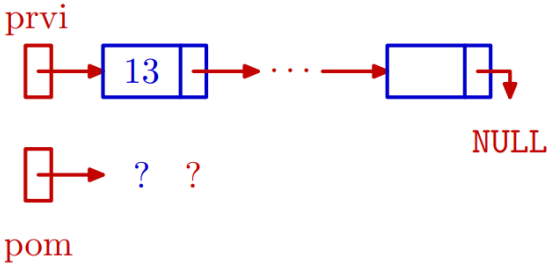
Obriši prvog




```
pom = prvi;
```




```
prvi = prvi->sljed;
```




```
free(pom);
```

Funkcija obrisi_prvog

```
lista obrisi_prvog(lista prvi){

    lista pom;

    if (prvi != NULL) {
        pom = prvi;
        prvi = prvi->sljed;
        free(pom);
       /* Ne treba pom = NULL; */
      }

    return prvi;
}
```

Kreiraj novi element i ubaci ga na početak

Operacije `kreiraj_novi` i `ubaci_na_pocetak` ima smisla spojiti u jednu operaciju `kreiraj_sprijeda`. Tada pomoćni pokazivač **novi** postaje **lokalni** objekt u funkciji.

```
1  lista kreiraj_sprijeda(lista prvi, int broj){
2
3        lista novi = NULL;
4        novi = (lista) malloc(sizeof(element));
5
6        if (novi == NULL) {
7           printf("Alokacija␣nije␣uspjela.\n");
8           exit(EXIT_FAILURE); /* exit(1); */
9         }
10
11       novi->broj = broj;
12       novi->sljed = prvi;
13       return novi; }
```

Funkcija obrisi_listu

Brisanje liste se može implementirati kao brisanje prvog elementa dok `prvi!=NULL`.

```
lista obrisi_listu(lista prvi){

     lista pom;

     while (prvi != NULL) {
            pom = prvi;
            prvi = prvi->sljed;
            free(pom);
       }

    return NULL; /* <=> return prvi; */
}
```

Funkcija broj_elemenata

Broj elemenata liste treba **izračunati**, kao i kod stringa, prolaskom do kraja liste.

```
int broj_elemenata(lista prvi){

  lista pom;
  int brojac = 0;

  for (pom = prvi; pom != NULL; pom = pom->sljed)
        ++brojac;

  return brojac;
}
```

Funkcija ispisi_listu

```
1  void ispisi_listu(lista prvi){
2
3    lista pom;
4    int brojac = 0;
5
6    for(pom = prvi; pom != NULL; pom = pom->sljed){
7            printf("␣Element␣%2d,␣broj␣=␣%2d\n",
8           ++brojac, pom->broj);
9      }
10
11   return;
12 }
```

Funkcija trazi_broj

Funkcija za traženje zadanog broja vraća pokazivač na prvi element koji sadrži zadani broj, ili NULL ako takvog elementa nema u listi. Uočite **skraćeno računanje** uvjeta u while petlji.

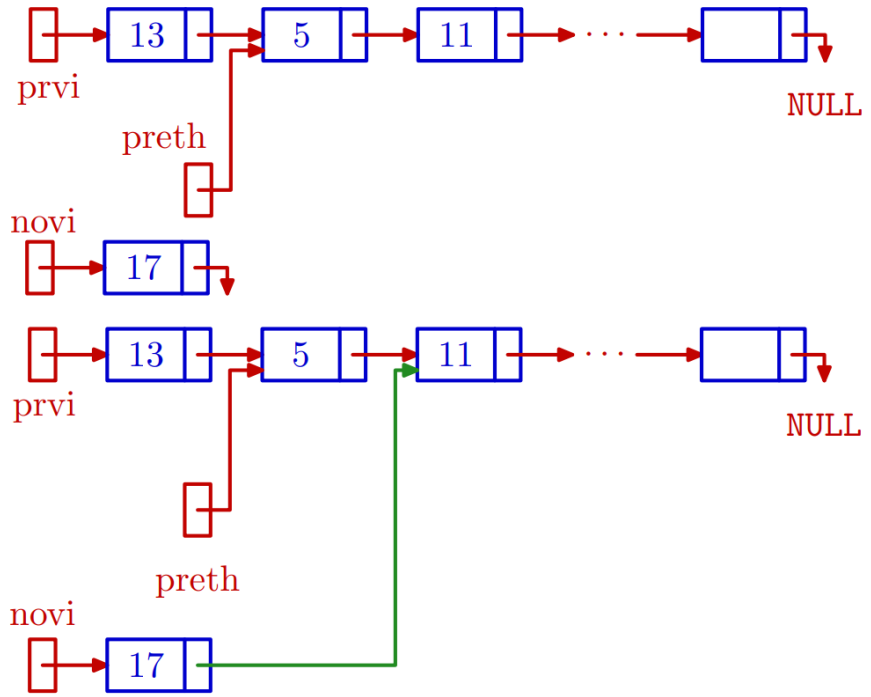
```
1  lista trazi_broj(lista prvi, int broj){
2
3       lista pom = prvi;
4
5       while (pom != NULL && pom->broj != broj)
6              pom = pom->sljed;
7     return pom;
8  }
```

Funkcija trazi_zadnji

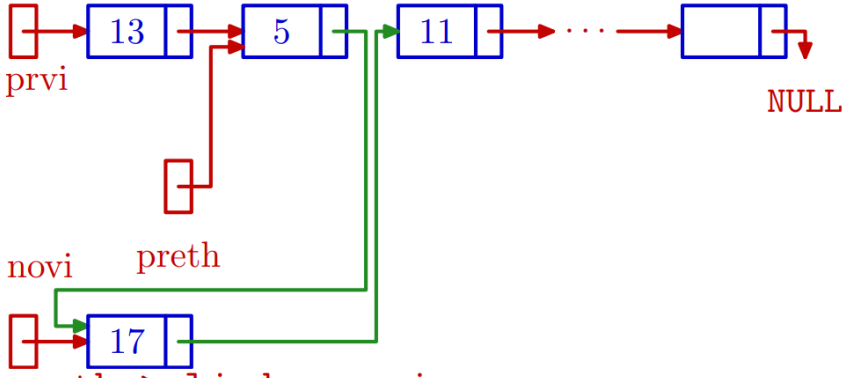
Funkcija vraća pokazivač na zadnji element u listi (njegov `sljed` je `NULL`), ili vraća `NULL` ako takvog elementa nema.

```
lista trazi_zadnji(lista prvi){

      lista pom;

      if (prvi == NULL) return NULL;

      for (pom = prvi; pom->sljed != NULL;
             pom = pom->sljed);

      return pom;
}
```

Ubaci bilo gdje iza prvog




```
novi->sljed = preth->sljed;
```

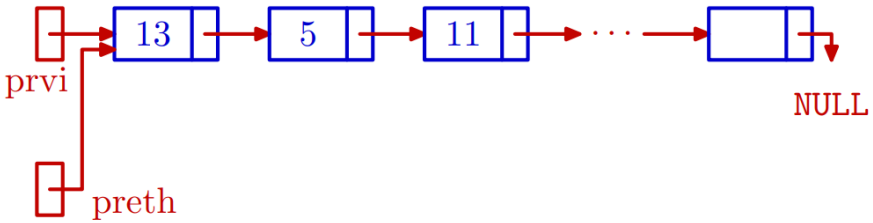


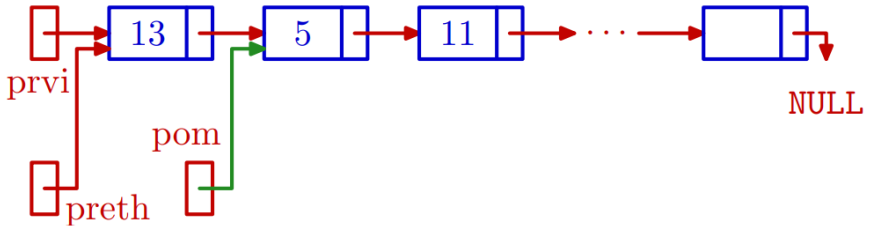

```
preth->sljed = novi;
```

Funkcija ubaci_iza

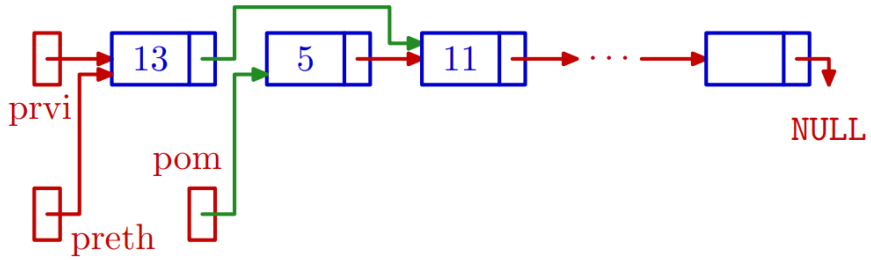
```
1  lista ubaci_iza(lista pr, lista pre, lista novi){
2    /* Ne provjerava novi != NULL. */
3    /* Ako je pre == NULL, ubacujemo na pocetak. */
4              if (pre == NULL) {
5                   novi->sljed = pr;
6                   pr = novi;
7                }
8              else {
9                   novi->sljed = pre->sljed;
10                  pre->sljed = novi;
11               }
12             return pr;
13 }
```

Obriši bilo gdje iza prvog






```
pom = preth->sljed;
```




```
preth->sljed = pom->sljed;
```

Ako izbačeni element (na koji pokazuje `pom`) zaista želimo obrisati, pozivamo `pom = obrisi_element(pom);` ili `free(pom);`
S tim elementom možemo raditi i druge operacije.

Funkcija obrisi_iza

```
lista obrisi_iza(lista prvi, lista preth){

        lista pom;
/* Ako je preth == NULL, brisemo prvi element. */
        if (preth == NULL) {
            pom = prvi;
            prvi = prvi->sljed;
           }
        else {
           pom = preth->sljed;
           preth->sljed = pom->sljed;
       }

       free(pom);
       return prvi;
}
```

Funkcija `obrisi_iza`

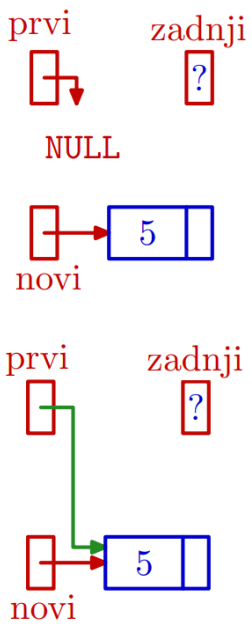
Implementacija funkcije ima dva nedostatka:

- ne provjerava je li ulazna lista prazna, tj. ne testira uvjet `prvi == NULL`
- ne pazi na kraj liste, slučaj `preth == zadnji`, tj. `preth->sljed == NULL`

Uz navedeno, pedantna varijanta provjerava pokazuje li `preth` na neki element liste zadane pokazivačem `prvi`.

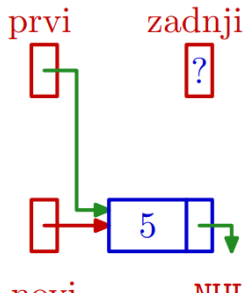
Popravite navedene nedostatke funkcije.

Ubaci na kraj s pamćenjem zadnjeg

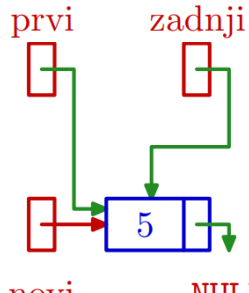



```
if (prvi == NULL)
    prvi = novi;
else    /* Ocekujemo zadnji->sljed == NULL. */
    zadnji->sljed = novi;
```

Ubaci na kraj s pamćenjem zadnjeg

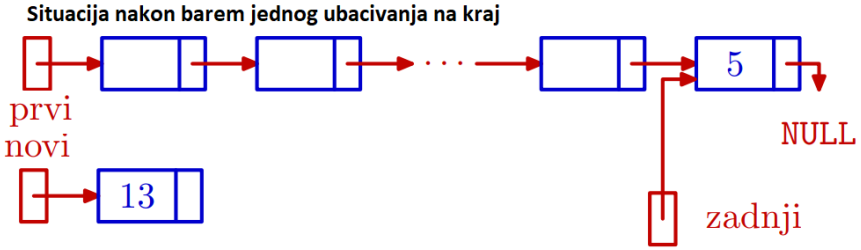



```
novi->sljed = NULL;
```

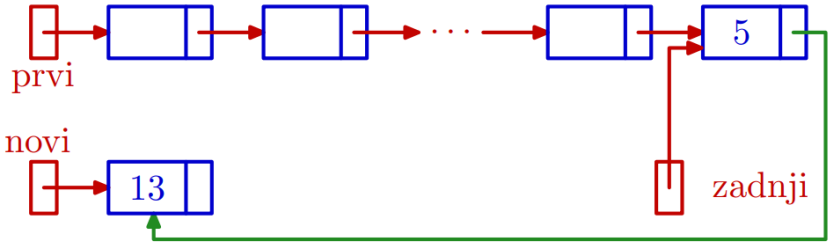



```
zadnji = novi;
```

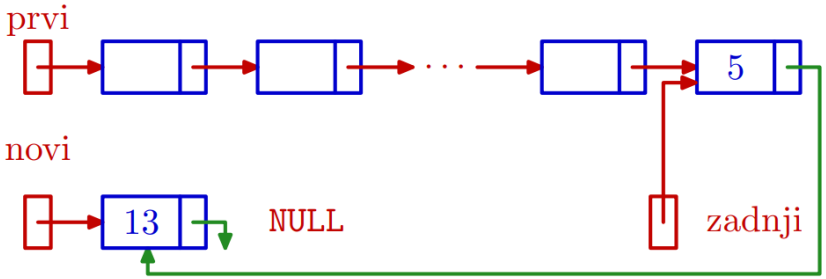
⋮



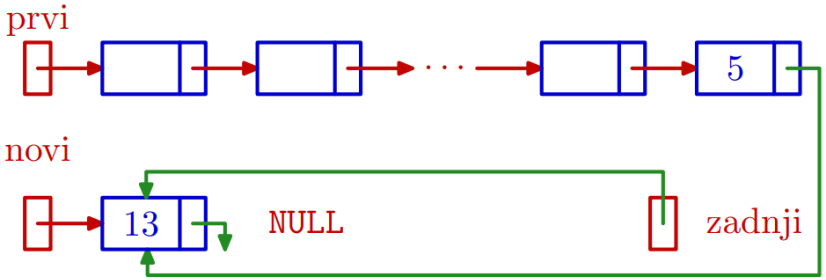
Ubaci na kraj s pamćenjem zadnjeg




```
if (prvi == NULL)
    prvi = novi;
else    /* Ocekujemo zadnji->sljed == NULL. */
    zadnji->sljed = novi;
```




```
novi->sljed = NULL;
```




```
zadnji = novi;
```

Funkcija ubaci_na_kraj

```
lista ubaci_na_kraj(lista prvi, lista *p_zadnji,
                                        lista novi){
/* Ne provjerava novi != NULL. */
/* Vraca zadnji kroz varijabilni
   argument - pokazivac p_zadnji. */

      lista zadnji = *p_zadnji;

     /* Moze: prvi == NULL || zadnji == NULL. */
      if (prvi == NULL)
           prvi = novi;
      else /* Ocekujemo zadnji->sljed == NULL. */
          zadnji->sljed = novi;
      novi->sljed = NULL;
/*Ne treba: zadnji = novi; *p_zadnji = zadnji;*/
      *p_zadnji = novi; /* Vrati novi zadnji! */
      return prvi; }
```

Funkcija kreiraj_straga

Napišite funkciju kreiraj_straga po ugledu na funkciju kreiraj_sprijeda sa zaglavljem:

```
lista kreiraj_straga(lista prvi, lista *p_zadnji,
                                          int broj)
```

koja kreira novi element za zadani broj, ubacuje ga na kraj liste i korektno vraća pokazivače na prvi i zadnji element.