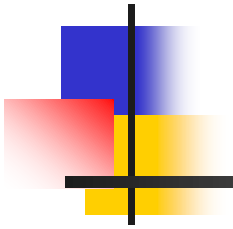
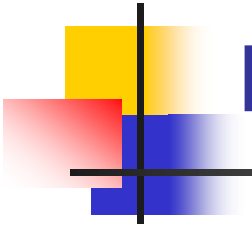


PROGRAMIRANJE II

P-06: Linearne strukture podataka

prof. dr **Dražen Brđanin**
2023/24



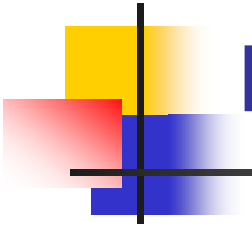
P-06: Linearne strukture podataka

- **Sadržaj predavanja**
  - Osnovni pojmovi o strukturama podataka
  - Osnovne linearne strukture
    - sekvencijalne
    - ulančane:
      - jednostruko povezana lista
      - dvostruko povezana lista
      - kružna lista
      - stek
      - red / kružni bafer

Osnovni pojmovi o strukturama podataka

program = algoritam + struktura podataka

- **Osnovni gradivni elementi za implementaciju programskih sistema**
  - **struktura podataka** (eng. *data structure*)**:** opisuje način organizacije podataka
  - **algoritam**: opisuje način obrade podataka
- **Struktura podataka:** modeluje objekte i podatke u problemu koji rješavamo
  - **elementarni/primitivni tipovi**
    - ugrađeni u programske jezike (reprezentacija i manipulacija elementarnim podacima)
  - **složeni/strukturirani**
    - nisu ugrađeni u programske jezike, nego korisnički definisani
    - složeni podatak je **kolekcija elementarnih podataka**
    - osim definicije strukture, **programer mora da implementira operacije za manipulaciju složenim podatkom** (primjena operacija nad elementarnim podacima)
- **Osnovni načini strukturiranja podataka**
  - **homogeno**: niz (kolekcija podataka istog tipa)
  - **heterogeno**: zapis /slog (zapis je kolekcija logički povezanih podataka različitih tipova)



Klasifikacije

- **Klasifikacija struktura prema vezama između elemenata:**
  - **linearna**:
    - kolekcija podataka u kojoj svaki element nema više od jednog neposrednog prethodnika niti više od jednog neposrednog sljedbenika
    - svaki element u kolekciji može biti povezan sa najviše druga dva elementa
  - **nelinearna**:
    - kolekcija podataka u kojoj elementi mogu da imaju više neposrednih prethodnika (predaka) i/ili više neposrednih sljedbenika (nasljednika)
    - svaki element u kolekciji može biti povezan sa više od druga dva elementa

- **Klasifikacija struktura prema mogućnosti promjene veličine:**
  - **statička**: inicijalno zadata i nepromjenljiva veličina kolekcije tokom životnog vijeka
  - **dinamička**: kolekcija može da mijenja veličinu tokom životnog vijeka (dodavanje, izbacivanje, ...)

- **Klasifikacija struktura prema mjestu čuvanja:**
  - **unutrašnja**: kolekcija se nalazi u operativnoj memoriji
  - **spoljašnja**: kolekcija se nalazi u sekundardnoj memoriji (datoteka)

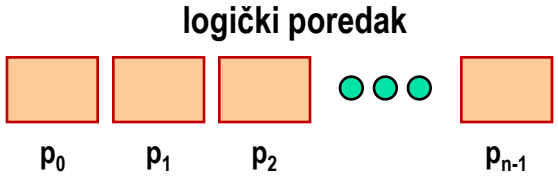
Memorijska reprezentacija

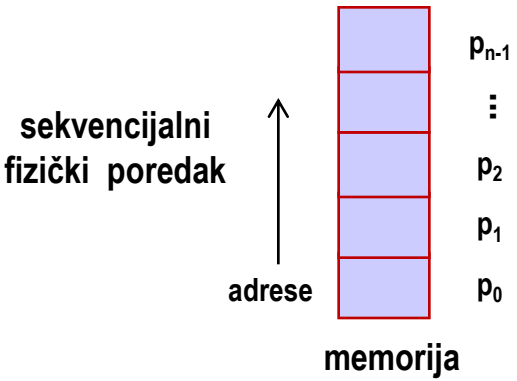
Sekvencijalna reprezentacija:

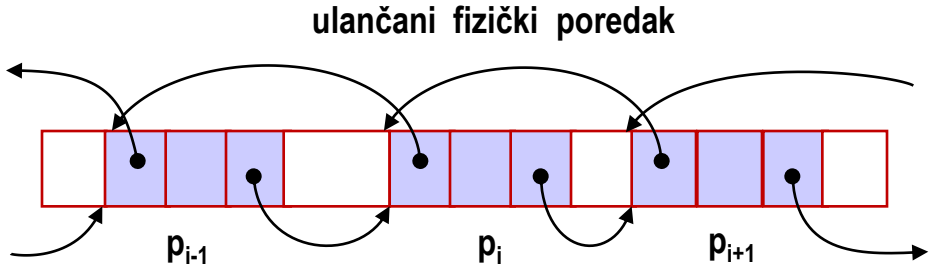
- **fizički i logički poredak elemenata u kolekciji je identičan**
- **pristup elementima je direktan** (moguće je direktno pristupiti svakom elementu kolekcije)
- uobičajeno se koristi za reprezentaciju linearnih struktura

Ulančana reprezentacija:

- **fizički i logički poredak elemenata u kolekciji se razlikuju**
- **pristup elementima je indirektan** (koristi se pokazivački mehanizam)
- uobičajeno se koristi za reprezentaciju nelinearnih struktura

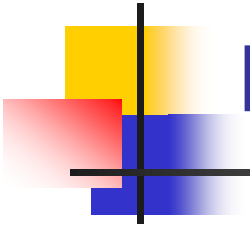






fizički poredak: razmještaj elemenata u memoriji

logički poredak: prethodnik↔sljedbenik / predak ↔potomak

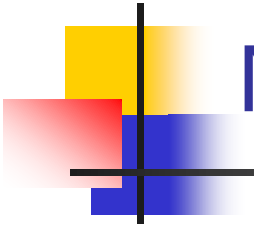


Linearne strukture podataka

- **Linearna struktura podataka**:
  - kolekcija podataka u kojoj svaki element nema više od jednog neposrednog prethodnika niti više od jednog neposrednog sljedbenika
  - svaki element u kolekciji može biti povezan sa najviše druga dva elementa
  - strukturno svojstvo: **JEDNODIMENZIONALNOST**
- **Implementacija:**
  - **sekvencijalna: niz** (statički, dinamički)
  - **ulančana: povezana (ulančana) lista** (jednostruko povezana, dvostruko povezana, kružna)
- **Operacije na linearnim strukturama:**
  - dodavanje
  - umetanje
  - brisanje
  - pretraživanje
  - pristup elementu
  - obilazak po poretku
  - sortiranje

Cijena (složenost) operacije zavisi od vrste implementacije strukture

Nizovi



- **Niz** = homogeni strukturirani tip
- sekvencijalna fizička reprezentacija
- **statički** niz
  - fiksna, unaprijed zadata veličina niza
  - ako kompajler ne podržava nizove promjenljive dužine, često imamo prostornu neiskorišćenost
- **dinamički** niz
  - veličina niza nije fiksna (može da se mijenja, ali cijena mijenjanja veličine može biti skupa, ako cijeli niz mora da se kopira)
  - bolja prostorna iskorišćenost, nego kod statičkih
- elementi su indeksirani (pomjeraj u odnosu na početak)
- višedimenzioni niz = niz nizova

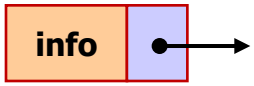
- **pristup** elementima niza
  - direktan (niz[i]) i **efikasan**
  - složenost pristupa $O(1)$
- **umetanje** (npr. održavanje uređenog poretka)
  - **neefikasno** (najgori slučaj: svi elementi se pomjeraju jedno mjesto prema kraju)
  - složenost umetanja $O(n)$
- **brisanje** (izbacivanje elementa)
  - **neefikasno** (najgori slučaj: svi se pomjeraju jedno mjesto prema početku)
  - složenost brisanja $O(n)$

Povezane (ulančane) liste

- **Povezana (ulančana) lista** (eng. *linked list*) = ulančana implementacija linearne strukture
- **dinamička struktura**
  - dinamička alokacija
  - pristup elementima je indirektan (pokazivači)
- osnovni element: **ČVOR** (eng. *node*)
  - informacioni sadržaj
  - pokazivač(i)

- **Prema načinu povezanosti**
  - jednostruko povezane liste
  - dvostruko povezane liste

Čvor u jednostruko povezanoj listi



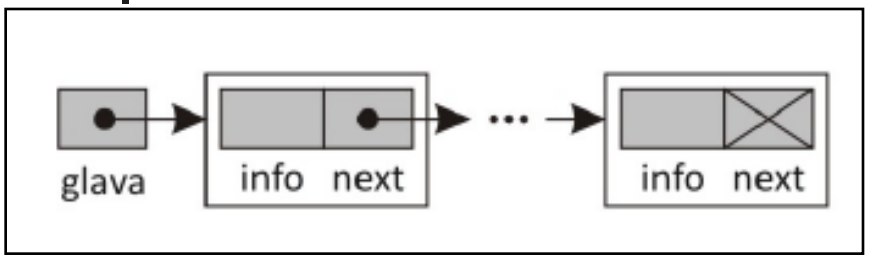
Čvor u dvostruko povezanoj listi

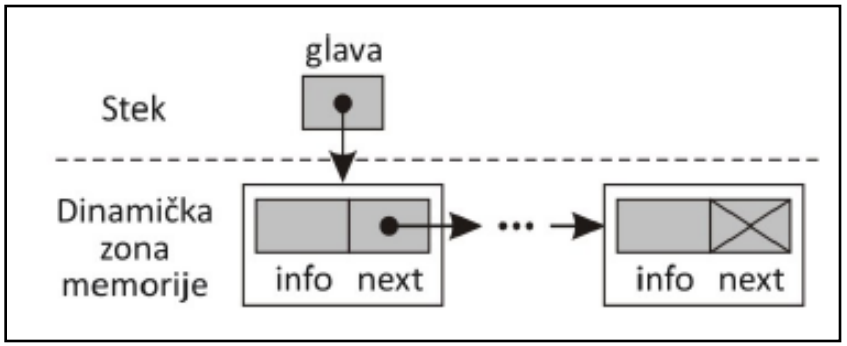

```
typedef struct node {
                      <tip> info;
                      struct node *next;
                    } NODE;
```

```
typedef struct node {
                         <tip> info;
                         struct node *left;
                         struct node *right;
                       } NODE;
```

Samoreferišuća struktura = struktura koja posjeduje pokazivač na istu strukturu

Jednostruko povezana lista





početak liste:

- pokazivač na prvi čvor
- uobičajeni nazivi (glava, head, list)
- na početku je lista prazna

čvorovi:

- čvorovi se dinamički alociraju kad treba da se doda novi element
- čvorovi se dinamički dealociraju kad se element briše

završetak/rep (eng. *tail*)**:**

- pokazivač u posljednjem čvoru ima vrijednost **NULL**

- **pristup** elementima jednostruko povezane liste
  - indirektan (preko pokazivača)
  - **neefikasno** (najgori slučaj: da bi se pristupilo posljednjem čvoru, mora da se redom prođe kroz sve čvorove – od glave do repa, jer se adresa sljedećeg čvora nalazi isključivo u prethodnom čvoru)
  - složenost pristupa $O(n)$
- **umetanje i brisanje**
  (npr. održavanje uređenog poretka)
  - **efikasnije nego kod nizova** – nema pomjeranja čvorova – samo se ažuriraju pokazivači

Jednostruko povezana lista

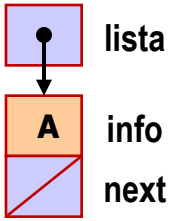
Formiranje liste:

- na početku je lista prazna

```
...
NODE *lista = NULL;
...
```

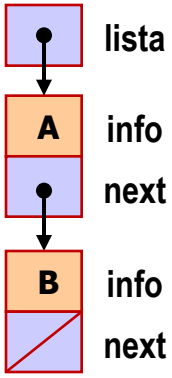
- dodavanje prvog čvora u listu

```
lista = (NODE *) malloc(sizeof(NODE));
lista->info = 'A';
lista->next = NULL;
...
```



- prolazak kroz listu i dodavanje novog čvora na kraj

```
NODE *tmp = lista;
while (tmp->next) tmp = tmp->next;
NODE *novi = (NODE *) malloc(sizeof(NODE));
novi->info = 'B';
novi->next = NULL;
tmp->next = novi;
```



Jednostruko povezana lista

Generalizovana procedura formiranja jednostruko povezane liste (dodavanje na kraj)

```
int formiranjeListe(NODE **lista, <tip> info)
{
   NODE *novi = (NODE *) malloc(sizeof(NODE));
   if (novi == NULL) return 0;
   novi->info = info;
   novi->next = NULL;

   if (*lista == NULL)
      *lista = novi;
   else
   {
      NODE *tmp = *lista;
      while (tmp->next)
         tmp = tmp->next;
      tmp->next = novi;
   }
   return 1;
}
```

lista = pokazivač na adresu pokazivača koji pokazuje na početak liste

novi = pokazivač na novi čvor koji se kreira u dinamičkoj zoni memorije

novi čvor se indirektno inicijalizuje (informacioni sadržaj + pointer na sljedeći čvor kojeg nema)

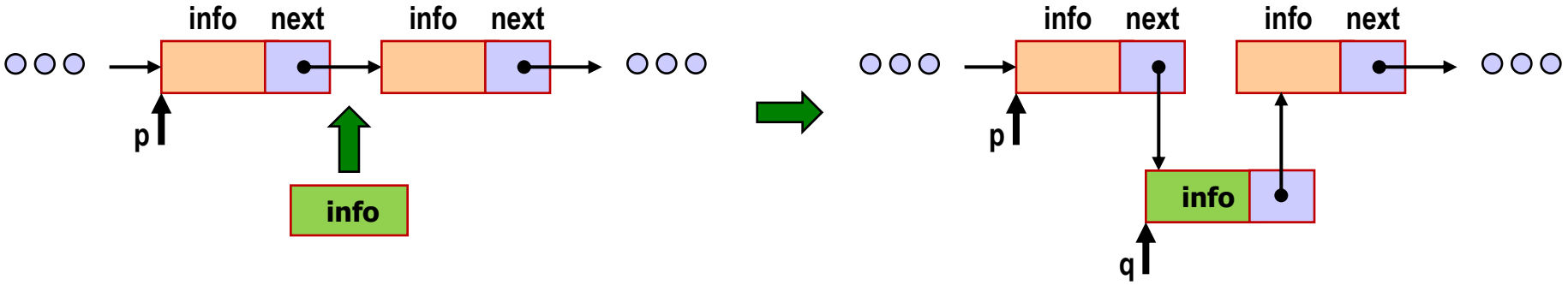
ako je lista prazna (*lista == NULL), treba dodati prvi čvor, tj. treba inicijalizovati pokazivač početka liste (indirektna inicijalizacija adresom novog čvora)

ako lista nije prazna, treba proći kroz listu do posljednjeg čvora (sve dok trenutni čvor ima sljedbenika)

na kraju se pokazivač u posljednjem čvoru setuje da pokazuje novi čvor

Jednostruko povezana lista

Ubacivanje čvora iza zadatog čvora



algoritam

```
insert_after(p, info)
{
   q = noviCvor(info)
   next(q) = next(p)
   next(p) = q
}
```

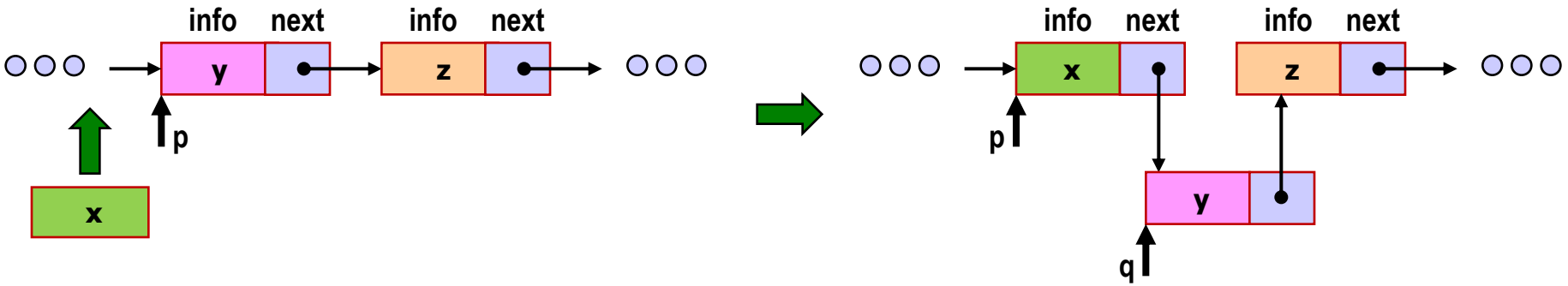
implementacija

```
int insert_after(NODE *p, <tip> info)
{
   NODE *q = (NODE *) malloc(sizeof(NODE));
   if (q==NULL) return 0;
   q->info = info;

   q->next = p->next;
   p->next = q;
   return 1;
}
```

Jednostruko povezana lista

Ubacivanje čvora ispred zadatog čvora



algoritam

```
insert_before(p, info)
{
   q = noviCvor(info(p))
   next(q) = next(p)
   next(p) = q
   info(p) = info
}
```

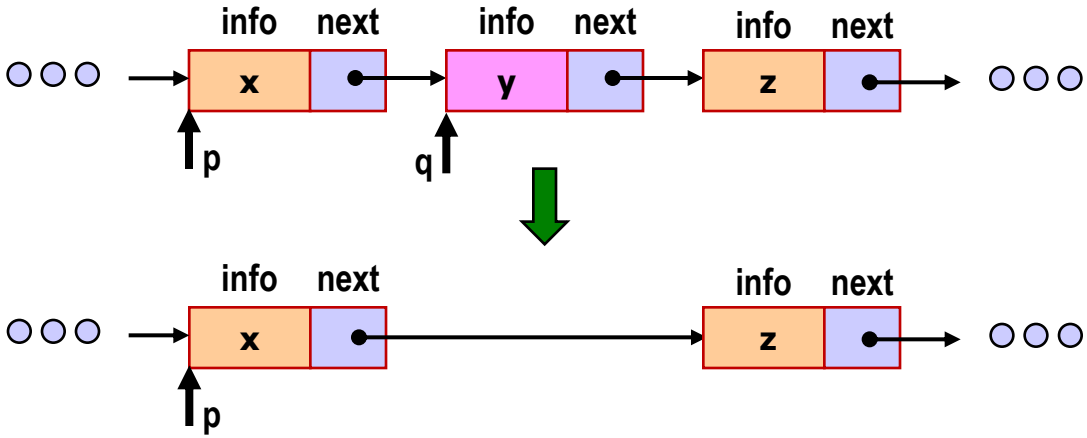
implementacija

```
int insert_before(NODE *p, <tip> info)
{
   NODE *q = (NODE *) malloc(sizeof(NODE));
   if (q==NULL) return 0;
   q->info = p->info;
   q->next = p->next;

   p->info = info;
   p->next = q;
   return 1;
}
```

Jednostruko povezana lista

Brisanje čvora iza zadatog čvora



algoritam

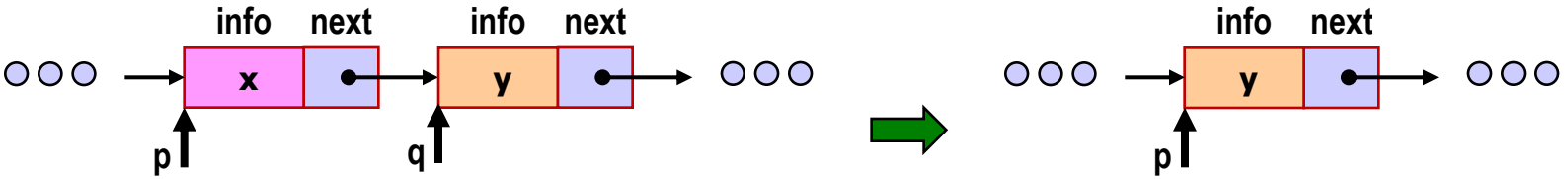
```
delete_after(p)
{
   q = next(p)
   next(p) = next(q)
   delete(q)
}
```

implementacija

```
int delete_after(NODE *p)
{
   if (!p) return 0;
   NODE *q = p->next;
   if (q)
   {
      p->next = q->next;
      free(q);
      return 1;
   }
   return 0;
}
```

Jednostruko povezana lista

Brisanje zadatog čvora



algoritam

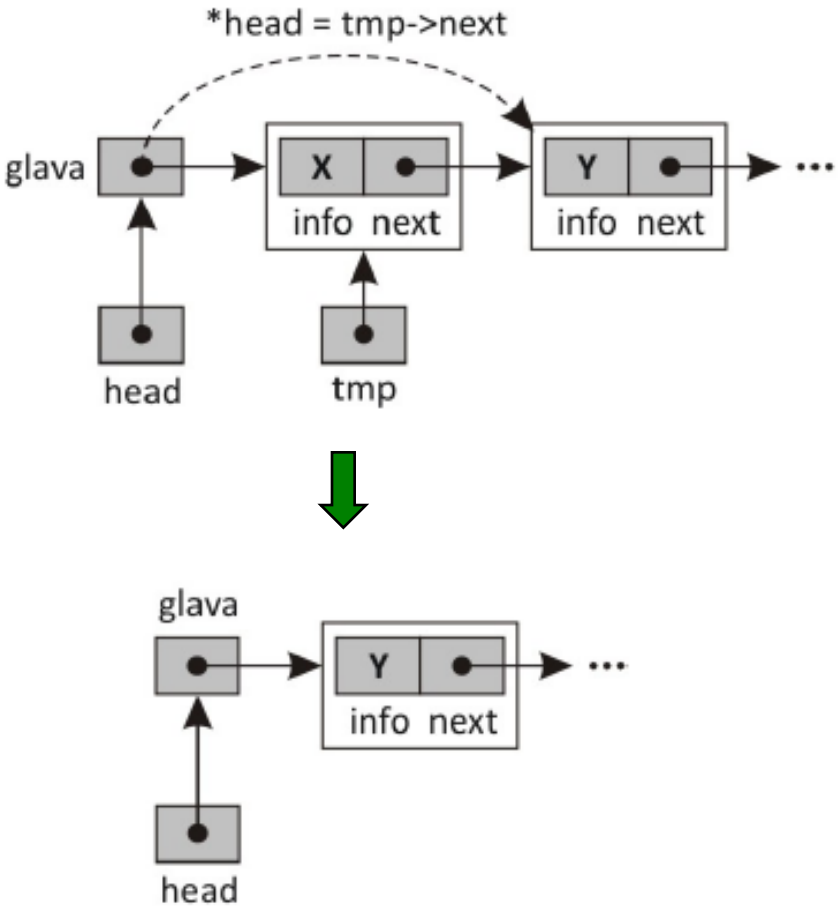
```
delete(p)
{
   q = next(p)
   info(p) = info(q)
   next(p) = next(q)
   delete(q)
}
```

implementacija

```
int delete_node(NODE *p)
{
   if (!p) return 0;
   NODE *q = p->next;
   if (q)
   {
     p->next = q->next;
     p->info = q->info;
     free(q);
     return 1;
   }
   return 0;
}
```

Jednostruko povezana lista

Brisanje početnog čvora




```
int deleteFront(NODE **head)
{
   if (*head == NULL) return 0;
   NODE *tmp = *head;
   *head = tmp->next;
   free(tmp);
   return 1;
}
```

Brisanje kompletne liste

```
void deleteList(NODE **head)
{
   NODE *tmp;
   while (*head)
   {
      tmp = *head;
      *head = tmp->next;
      free(tmp);
   }
}
```

Jednostruko povezana lista

Pretraživanje neuređene liste

Informacioni sadržaj u čvorovima nije uređen, pa je neophodno pretraživanje cjelokupne liste

Lista se pretražuje sve dok pokazivač na tekući nije NULL i dok je informacioni sadržaj (ključ) različit od ključa pretrage

Po izlasku iz petlje, ili je dostignut kraj ili je pronađen traženi čvor – dovoljno je vratiti vrijednost koju pokazuje pokazivač trenutne pozicije u listi

```
NODE *search(NODE *lista, <tip> info)
{
   while (lista && lista->info != info)
      lista = lista->next;
   return lista;
}
```

Pretraživanje uređene liste

Informacioni sadržaj u čvorovima je uređen, pa nije neophodno pretraživanje cjelokupne liste

Lista se pretražuje sve dok pokazivač na tekući nije NULL i dok je informacioni sadržaj manji od ključa pretrage

Po izlasku iz petlje:

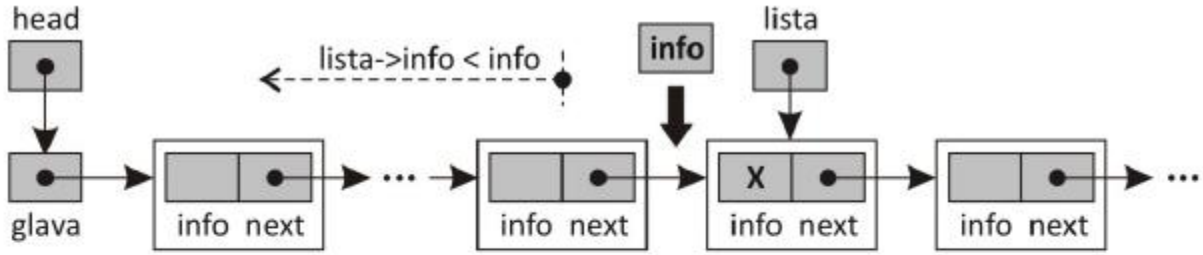
- ako je dostignut kraj ili ako je informacioni sadržaj veći od ključa pretrage – nije pronađen traženi čvor pa treba vratiti NULL
- Inače je pronađen traženi čvor i treba vrijednost pokazivača trenutne pozicije u listi

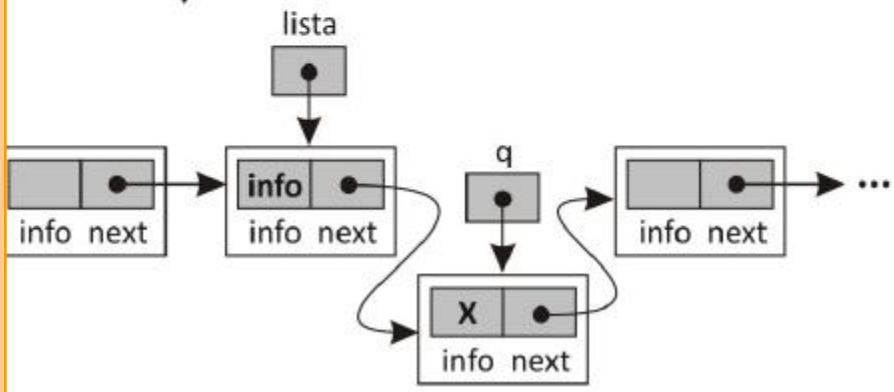
```
NODE *search_sort(NODE *lista, <tip> info)
{
   while (lista && lista->info < info)
      lista = lista->next;
   if (!lista || lista->info > info)
      return NULL;
   return lista;
}
```

Jednostruko povezana lista

Sortiranje liste

(formiranje sortirane liste)



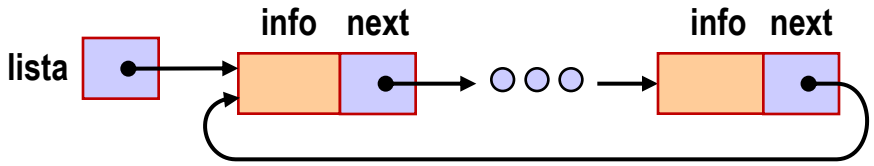



```
int formirajSortList(NODE **head, INFO info)
{
  if (*head == NULL)
  {
     *head = noviCvor(info));
     return *head != NULL;
  }
  NODE *lista = *head;
  while (lista->next && lista->info<info)
     lista = lista->next;
  if (lista->info < info)
     return insertAfter(lista,info);
  else
     return insertBefore(lista,info);
}
```

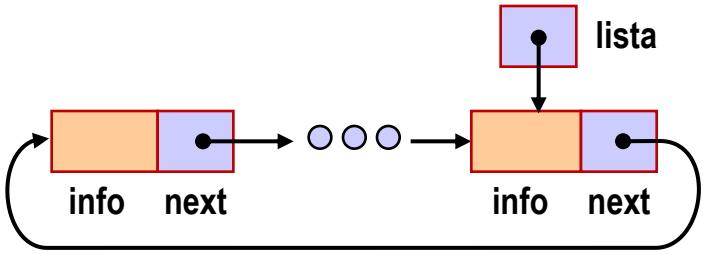
Jednostruko povezana kružna lista

Jednostruko povezana kružna lista

- pokazivač u posljednjem čvoru pokazuje na prvi čvor, što omogućava kružno kretanje kroz listu

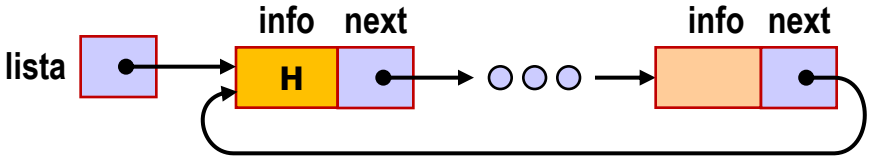


- spoljašnji pokazivač na listu ponekad pokazuje na posljednji čvor (a ne na prvi)



Kružna lista sa zaglavljem

- na početku liste se nalazi poseban čvor – **zaglavlje**
- zaglavlje se uglavnom koristi da označi prvi čvor u listi (ako se spoljašnji pokazivač pomjera, npr. ako spoljašnji pokazivač pokazuje na posljednji čvor)



- zaglavlje se nikad ne briše, pa lista nikad nije prazna