

## Dvostruko spregnuta lista

```
/**
 * Čvor dvostruko spregnute liste.
 * Svaki čvor liste u sebi čuva podatak (kod ovog čvora to je ceo broj)
 * i pokazivače na prethodni i sledeći čvor.
 *
 * @author Kostandin Dimitrijević, Strukture podataka i algoritmi, FON, 2010
 */
public class CvorDSListe {
    // podatak koji se čuva u čvoru.
    // Ako čvor treba da čuva neki drugi tip elementa, ovde to treba promeniti
    int podatak;

    // "pokazivač" na prethodni čvor u listi
    CvorDSListe prethodni;

    // "pokazivač" na sledeći čvor u listi
    CvorDSListe sledeci;

    /**
     * Konstruktor. Prihvata i postavlja sve elemente čvora.
     * @param p Podatak koji se čuva u čvoru.
     * @param pret Pokazivač na prethodni element liste
     * @param sled Pokazivač na sledeći element za čvor koji se kreira
     */
    public CvorDSListe(int p, CvorDSListe pret, CvorDSListe sled){
        podatak = p;
        prethodni = pret;
        sledeci = sled;
    }
}

package LinearneStrukture;

/**
 * Dvostruko spregnuta lista. Dinamička opšta lista - elementi mogu da se
 * dodaju bilo gde u strukturu i bilo koji element može da se izbaci.
 * Redosled ubacivanja i izbacivanja nije definisan. Elementi su uvezani preko
 * dve veze "unapred" i "unazad", tako da je moguće direktno kretanje kroz strukturu
 * u smeru od prvog do poslednjeg i od poslednjeg do prvog. Na svaki čvor pokazuje
 * tačno dva pokazivača, te je pri ubacivanju novog čvora potrebno povezati
 * dva pokazivača na njega, dok je pri izbacivanju potrebno razvezati dva
 * pokazivača. Gledano sa oba kraja, lista je strukturno potpuno simetrična.
 *
 * @author Kostandin Dimitrijević, Strukture podataka i algoritmi, FON, 2010
 */
public class DSLista {
    // Pokazivač na prvi čvor u listi.
    private CvorDSListe prvi;

    // Pokazivač na poslednji čvor u listi.
    private CvorDSListe poslednji
    private DSLista(){
        prvi = poslednji = null;
    }

    /**
     * Proverava da li je lista prazna. Lista je prazna kada ne postoji
     * pokazivač na prvi čvor.
     * @return true ako je lista prazna. u suprotnom false
     */
    public boolean praznaLista(){
        return prvi == null;
    }
}
```

```

/**
 * Ubacivanje novog elementa na početak liste.
 * @param podatak Podatak koji će se čuvati u novom čvoru
 */
public void ubaciNaPocetak(int podatak){
    // Kreiranje novog čvora. Pošto se novi čvor kreira na početku
    // strukture, on nema prethodnika, a njegov sledeći element će biti
    // trenutno prvi element.
    CvorDSListe novi = new CvorDSListe(podatak, null, prvi);

    // ako je lista prazna, onda je novokreirani element i poslednji element
    if (praznaLista())
        poslednji = novi;
    else
        // ako lista nije prazna onda je novokreirani element prethodnik
        // trenutno prvom elementu
        prvi.prethodni = novi;
    // nakon povezivanja treba postaviti da je novi element prvi
    prvi = novi;
}
/**
 * Ubacivanje novog elementa na kraj liste.
 * @param podatak Podatak koji će se čuvati u novom čvoru
 */
public void ubaciNaKraj(int podatak){
    // Kreiranje novog čvora. Pošto se novi čvor kreira na kraju
    // strukture, on nema sledbenika, a njegov prethodnik će biti
    // trenutno poslednji element.
    CvorDSListe novi = new CvorDSListe(podatak, poslednji, null);

    // ako je lista prazna, onda je novokreirani element i prvi element
    if (praznaLista())
        prvi = novi;
    else
        // ako lista nije prazna onda je novokreirani element sledbenik
        // trenutno poslednjem elementu
        poslednji.sledeci = novi;
    // nakon povezivanja treba postaviti da je novi element prvi
    poslednji = novi;
}
/**
 * Izbacivanje elementa sa počtka liste. Element se izbacuje tako što se
 * pokazivač na prvi čvor preveže da pokazuje na drugi čvor i razveže
 * veza "unazad" između drugog i prvog čvora. Time niko više
 * ne pokazuje na taj čvor čime se on briše iz memorije.
 *
 * @return Vrednost iz čvora koji je izbačen.
 * @throws Exception Ako je lista prazna baca se izuzetak.
 */
public int izbaciSaPocetka() throws Exception{
    // Ako je lista prazna baca se izuzetak
    if (praznaLista())
        throw new Exception("lista je prazna");

    // sačuvamo podatak iz prvog čvora u pomoćnu promenljivu
    int podatak = prvi.podatak;
    // prebacujemo prvi na drugi čvor
    prvi = prvi.sledeci;
    // ako je lista imala samo jedan element, onda je sada prvi == null,
    // pa treba postaviti i da je poslednji == null
    if (prvi == null)
        poslednji = null;
    else
        // razvezivanje veze unazad
        prvi.prethodni = null;
    return podatak;
}

```

```

/**
 * Izbacivanje elementa sa kraja liste. Element se izbacuje tako što se
 * pokazivač na poslednji čvor preveže da pokazuje na pretposlednji čvor
 * i razveže veza "inapred" između pretposlednjeg i poslednjeg čvora.
 * Time niko više ne pokazuje na taj čvor čime se on briše iz memorije.
 *
 * @return Vrednost iz čvora koji je izbačen.
 * @throws Exception Ako je lista prazna baca se izuzetak.
 */
public int izbaciSaKraja() throws Exception{
    // Ako je lista prazna baca se izuzetak
    if (praznaLista())
        throw new Exception("lista je prazna");

    // sačuvamo podatak iz poslednjeg čvora u pomoćnu promenljivu
    int podatak = poslednji.podatak;
    // prebacujemo poslednji na pretposlednji čvor
    poslednji = poslednji.prethodni;
    // ako je lista imala samo jedan element, onda je sada poslednji == null, pa treba
    postaviti i da je prvi == null
    if (poslednji == null)
        prvi = null;
    else
        // razvezivanje veze unapred
        poslednji.sledeci = null;
    return podatak; }

/**
 * Ubacivanje elementa u sortiranu listu tako da ona ostane sortirana u
 * rastućem redosledu. Prolazi se kroz listu i traži onaj čvor čiji je
 * podatak veći od elementa koji se ubacuje. Nakon toga se novi čvor
 * ubacuje između trenutnog i njegovog prethodnika.
 *
 * @param podatak Podatak koji će se čuvati u novom čvoru
 */
public void ubaciUSort(int podatak){
    // Ako je lista prazna ili ako je podatak u poslenjem čvoru nije manji od
    // podatka koji se ubacuje, onda se vrši ubacivanje na kraj liste.
    if (praznaLista() || poslednji.podatak <= podatak)
        ubaciNaKraj(podatak);
    else {
        CvorDSListe pom = prvi;
        // tražimo element čiji je podatak veći od elementa koji ubacujemo
        while(pom.podatak < podatak){
            pom = pom.sledeci;
        }
        // umetanje novog čvora
        CvorDSListe novi = new CvorDSListe(podatak, pom.prethodni, pom);
        pom.prethodni = novi;
        if (novi.prethodni == null)
            prvi = novi;
        else
            novi.prethodni.sledeci = novi;
    }
}
}

```