

Osnovi programiranja
Beleške sa vežbi
Drugi semestar školske 2005/2006 godine

Smer *Računarstvo i informatika*
Matematički fakultet, Beograd

Jelena Tomašević

May 15, 2006

Sadržaj

1	Programski jezik C	5
2	Programski jezik C	7
2.1	Liste	7
2.1.1	Dvosturko povezana kružna lista	16
2.2	Zadaci za vježbu	18

1

Programski jezik C

1

¹Zasnovano na primerima sa sajta <http://www.matf.bg.ac.yu/~filip>

2

Programski jezik C

2.1 Liste

Primer 1 *Ubacivanje na početak jednostruko povezane liste - verzija sa **. Ispis i oslobađanje liste realizovani iterativno.*

```
#include <stdio.h>
#include <stdlib.h>

typedef struct cvor
{
    int br;
    struct cvor* sl;
} CVOR;

/* Pomocna funkcija koja kreira cvor liste sa datim sadrzajem.
   Funkcija kreira cvor i postavlja mu sadrzaj na dati broj.
   Polje sl ostaje nedefinisano.
   Funkcija vraca pokazivac na kreirani cvor. */

CVOR* napravi_cvor(int br)
{
    CVOR* novi = (CVOR*)malloc(sizeof(CVOR));
    if (novi == NULL)
    {
        fprintf(stderr, "Greska prilikom alokacije memorije\n");
        exit(1);
    }
    novi->br = br;
    return novi;
}

/* Zbog prenosa po vrednosti, sledeca funkcija ne radi ispravno */
/*
void ubaci_na_pocetak(CVOR* l, int br)
{
    CVOR* novi = napravi_cvor(br);
    novi->sl = l;
```

```
    l = novi; /* Ovde se menja lokalna kopija pokazivaca l, a
              ne l iz funkcije pozivaoca (main) */
}
*/

/* Ubacuje dati broj na pocetak liste.
   Pokazivac na pocetak liste se prenosi preko pokazivaca, umesto po
   vrednosti, kako bi mogla da mu se izmeni vrednost. */
void ubaci_na_pocetak(CVOR** pl, int br)
{
    CVOR* novi = napravi_cvor(br);
    novi->sl = *pl;
    *pl = novi;
}

/* Ispisivanje liste : iterativna verzija */
void ispisi_listu(CVOR* l)
{
    CVOR* t;
    for (t = l; t != NULL; t=t->sl)
        printf("%d ", t->br);
}

/* Sledeca funkcija je neispravna */
/*
void oslobodi_listu(CVOR* l)
{
    CVOR* t;
    for (t = l; t!=NULL; t = t->sl)
        free(t);
    /* Ovde se unistava sadrzaj cvora na koji ukazuje t.
       Korak petlje t = t->sl nece moci da se izvrši */
}
*/

/* Oslobadjanje liste : iterativna verzija */
void oslobodi_listu(CVOR* l)
{
    while (l)
    {
        CVOR* tmp = l->sl;
        free(l);
        l = tmp;
    }
}

main()
{
    CVOR* l = NULL;
```

```
    int i;
    for (i = 0; i<10; i++)
        ubaci_na_pocetak(&l, i);

    ispisi_listu(l);
    putchar('\n');

    oslobodi_listu(l);
}
```

Primer 2 Ubacivanje na početak jednostruko povezane liste - verzija sa eksplicitnim vraćanjem novog početka liste. Ispis i oslobađanje liste su realizovani rekursivno.

```
#include <stdio.h>
#include <stdlib.h>

typedef struct cvor
{
    int br;
    struct cvor* sl;
} CVOR;

/* Pomocna funkcija koja kreira cvor liste sa datim sadrzajem.
   Funkcija kreira cvor i postavlja mu sadrzaj na dati broj.
   Polje sl ostaje nedefinisano.
   Funkcija vraca pokazivac na kreirani cvor. */

CVOR* napravi_cvor(int br)
{
    CVOR* novi = (CVOR*)malloc(sizeof(CVOR));
    if (novi == NULL)
    {
        fprintf(stderr, "Greska prilikom alokacije memorije\n");
        exit(1);
    }
    novi->br = br;
    return novi;
}

/* Ubacuje dati broj na pocetak date liste.
   Funkcija pozivaocu eksplicitno vraca pocetak rezultujuce liste.*/
CVOR* ubaci_na_pocetak(CVOR* l, int br)
{
    CVOR* novi = napravi_cvor(br);
    novi->sl = l;
    return novi;
}

/* Ispisivanje liste : rekursivna verzija */
void ispisi_listu(CVOR* l)
{
    if (l != NULL)
```

```

    {
        printf("%d ", l->br);
        ispisi_listu(l->sl);
    }
}

/* Ispisivanje liste unatrag : rekurzivna verzija */
/* Prethodna funkcija se lako modifikuje tako da ispisuje listu unazad */
void ispisi_listu_unazad(CVOR* l)
{
    if (l != NULL)
    {
        ispisi_listu_unazad(l->sl);
        printf("%d ", l->br);
    }
}

/* Oslobadjanje liste : rekurzivna verzija */
void oslobodi_listu(CVOR* l)
{
    if (l != NULL)
    {
        oslobodi_listu(l->sl);
        /* Prvo se oslobadja poslednji element liste */
        /* printf("Oslobadjam %d\n", l->br); */
        free(l);
    }
}

main()
{
    CVOR* l = NULL;
    int i;
    for (i = 0; i<10; i++)
        l = ubaci_na_pocetak(l, i);

    ispisi_listu(l);
    putchar('\n');

    ispisi_listu_unazad(l);
    putchar('\n');

    oslobodi_listu(l);
}

```

Primer 3 Ubacivanje na kraj jednostruko povezane liste - verzija sa ** - iterativna i rekurzivna verzija

```

#include <stdio.h>
#include <stdlib.h>

typedef struct cvor

```

```
{
    int br;
    struct cvor* sl;
} CVOR;

/* Pomocna funkcija koja kreira cvor liste sa datim sadrzajem.
   Funkcija kreira cvor i postavlja mu sadrzaj na dati broj.
   Polje sl ostaje nedefinisano.
   Funkcija vraca pokazivac na kreirani cvor. */

CVOR* napravi_cvor(int br)
{
    CVOR* novi = (CVOR*)malloc(sizeof(CVOR));
    if (novi == NULL)
    {
        fprintf(stderr, "Greska prilikom alokacije memorije\n");
        exit(1);
    }
    novi->br = br;
    return novi;
}

/* Ubacuje dati broj na pocetak liste.
   Pokazivac na pocetak liste se prenosi preko pokazivaca, umesto po
   vrednosti, kako bi mogla da mu se izmeni vrednost.
   Iterativna verzija funkcije */
/* Ubacivanje na kraj liste je neefikasna operacija */
void ubaci_na_kraj(CVOR** pl, int br)
{
    CVOR* novi = napravi_cvor(br);
    novi->sl = 0;

    if (*pl == NULL)
        *pl = novi;
    else
    {
        /* Pronalazimo poslednji element liste - t*/
        CVOR* t;
        for (t=*pl; t->sl!=NULL; t=t->sl)
            ;
        t->sl = novi;
    }
}

/* Rekurzivna varijanta prethodne funkcije */
void ubaci_na_kraj_rekurzivno(CVOR** pl, int br)
{
    if (*pl == NULL)
    {
        CVOR* novi = napravi_cvor(br);
        *pl = novi;
    }
}
```

```

    }
    else
        ubaci_na_kraj_rekurzivno( &((*pl)->sl) ,br);
}

/* Ispisivanje liste : iterativna verzija */
void ispisi_listu(CVOR* l)
{
    CVOR* t;
    for (t = l; t != NULL; t=t->sl)
        printf("%d ", t->br);
}

/* Iterativna verzija funkcije koja oslobadja listu */
void oslobodi_listu(CVOR* l)
{
    while (l)
    {
        CVOR* tmp = l->sl;
        free(l);
        l = tmp;
    }
}

main()
{
    CVOR* l = NULL;
    int i;
    for (i = 0; i<5; i++)
        ubaci_na_kraj(&l, i);
    for (; i<10; i++)
        ubaci_na_kraj_rekurzivno(&l, i);

    ispisi_listu(l);
    putchar('\n');

    oslobodi_listu(l);
}

```

Primer 4 *Ubacivanje na kraj jednostruko povezane liste - verzija sa eksplicitnim vraćanjem nove liste - iterativna i rekurzivna verzija.*

```

#include <stdio.h>
#include <stdlib.h>

typedef struct cvor
{
    int br;
    struct cvor* sl;
} CVOR;

/* Pomocna funkcija koja kreira cvor liste sa datim sadrzajem.

```

Funkcija kreira cvor i postavlja mu sadrzaj na dati broj.
Polje sl ostaje nedefinisano.
Funkcija vraca pokazivac na kreirani cvor. */

```
CVOR* napravi_cvor(int br)
{
    CVOR* novi = (CVOR*)malloc(sizeof(CVOR));
    if (novi == NULL)
    {
        fprintf(stderr, "Greska prilikom alokacije memorije\n");
        exit(1);
    }
    novi->br = br;
    return novi;
}

/* Funkcija vraca pocetak rezultujuce liste */
CVOR* ubaci_na_kraj(CVOR* l, int br)
{
    CVOR* novi = napravi_cvor(br);
    novi->sl = NULL;

    if (l == NULL)
        return novi;
    else
    {
        CVOR* t;
        for (t = l; t->sl!=NULL; t=t->sl)
            ;
        t->sl = novi;

        /* Pocetak se nije promenio */
        return l;
    }
}

/* Rekurzivna varijanta prethodne funkcije.
   I ova funkcija vraca pokazivac na pocetak rezultujuce liste */
CVOR* ubaci_na_kraj_rekurzivno(CVOR* l, int br)
{
    if (l == NULL)
    {
        CVOR* novi = napravi_cvor(br);
        return novi;
    }

    l->sl = ubaci_na_kraj_rekurzivno(l->sl, br);
    return l;
}

/* Ispisivanje liste : iterativna verzija */
void ispisi_listu(CVOR* l)
```

```

{
    CVOR* t;
    for (t = l; t != NULL; t=t->sl)
        printf("%d ", t->br);
}

/* Iterativna verzija funkcije koja oslobadja listu */
void oslobodi_listu(CVOR* l)
{
    while (l)
    {
        CVOR* tmp = l->sl;
        free(l);
        l = tmp;
    }
}

main()
{
    CVOR* l = NULL;
    int i;
    for (i = 0; i<5; i++)
        l = ubaci_na_kraj(l, i);
    for (; i<10; i++)
        l = ubaci_na_kraj_rekurzivno(l, i);

    ispisi_listu(l);
    putchar('\n');

    oslobodi_listu(l);
}

```

Primer 5 Ubacivanje na odgovarajuće mesto sortirane jednostruko povezane liste - verzija sa **
- iterativna i rekurzivna verzija

```

#include <stdio.h>
#include <stdlib.h>

typedef struct cvor
{
    int br;
    struct cvor* sl;
} CVOR;

/* Pomocna funkcija koja kreira cvor liste sa datim sadrzajem.
   Funkcija kreira cvor i postavlja mu sadrzaj na dati broj.
   Polje sl ostaje nedefinisano.
   Funkcija vraca pokazivac na kreirani cvor. */

CVOR* napravi_cvor(int br)
{
    CVOR* novi = (CVOR*)malloc(sizeof(CVOR));

```

```
    if (novi == NULL)
    {
        fprintf(stderr, "Greska prilikom alokacije memorije\n");
        exit(1);
    }
    novi->br = br;
    return novi;
}
/* Ključna ideja u realizaciji ove funkcije je pronalazanje poslednjeg
   elementa liste čiji je ključ manji od datog elementa br.
*/
void ubaci_sortirano(CVOR** pl, int br)
{
    CVOR* novi = napravi_cvor(br);

    /* U sledeća dva slučaja ne postoji cvor čiji je ključ manji
       od datog broja (br)
       - Prvi je slučaj prazne liste
       - Drugi je slučaj kada je br manji od prvog elementa

       U oba slučaja ubacujemo na početak liste.
    */
    if (*pl == NULL || br < (*pl)->br)
    {
        novi->sl = *pl;
        *pl = novi;
        return;
    }

    /* Krenemo od početka i idemo dalje sve dok t nije poslednji
       manji element liste ili eventualno bas poslednji */
    CVOR* t;
    for (t = *pl; t->sl!=NULL && t->sl->br < br; t=t->sl)
        ;
    novi->sl = t->sl;
    t->sl = novi;
}

/* Rekurzivna verzija prethodne funkcije */
void ubaci_sortirano_rekurzivno(CVOR** pl, int br)
{
    if (*pl == NULL || br < (*pl)->br)
    {
        CVOR* novi = napravi_cvor(br);
        novi->sl = *pl;
        *pl = novi;
        return;
    }

    ubaci_sortirano(&((*pl)->sl), br);
}
```

```
/* Ispisivanje liste : iterativna verzija */
void ispisi_listu(CVOR* l)
{
    CVOR* t;
    for (t = l; t != NULL; t=t->sl)
        printf("%d ", t->br);
}

/* Iterativna verzija funkcije koja oslobadja listu */
void oslobodi_listu(CVOR* l)
{
    while (l)
    {
        CVOR* tmp = l->sl;
        free(l);
        l = tmp;
    }
}

main()
{
    CVOR* l = NULL;
    CVOR* k = NULL;
    int i;

    ubaci_sortirano(&l, 5);
    ubaci_sortirano(&l, 8);
    ubaci_sortirano(&l, 7);
    ubaci_sortirano(&l, 6);
    ubaci_sortirano(&l, 4);

    ubaci_sortirano_rekurzivno(&k, 5);
    ubaci_sortirano_rekurzivno(&k, 8);
    ubaci_sortirano_rekurzivno(&k, 7);
    ubaci_sortirano_rekurzivno(&k, 6);
    ubaci_sortirano_rekurzivno(&k, 4);

    ispisi_listu(l);
    putchar('\n');

    ispisi_listu(k);
    putchar('\n');

    oslobodi_listu(l);
}
```

2.1.1 Dvosturko povezana kružna lista

Primer 6 *Napisati funkciju koja omogućava umetanje čvora u dvostruko povezanu kružnu listu kao i izbacivanje čvora iz dvostruko povezane kružne liste. Omogućiti i štampanje podataka koje čuva lista.*

```
/* Program implementira deciju razbrajalicu eci-peci-pec i služi  
da ilustruje rad sa dvostruko povezanim kruznim listama */
```

```
#include <stdlib.h>
```

```
#include <stdio.h>
```

```
/* Dvostruko povezana lista */
```

```
typedef struct _cvor
```

```
{
```

```
    int broj;
```

```
    struct _cvor* prethodni, *sledeci;
```

```
} cvor;
```

```
/* Umetanje u dvostruko povezanu listu */
```

```
cvor* ubaci(int br, cvor* lista)
```

```
{
```

```
    cvor* novi=(cvor*)malloc(sizeof(cvor));
```

```
    if (novi==NULL)
```

```
    {    printf("Greska prilikom alokacije memorije \n");  
        exit(1);
```

```
    }
```

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

```
    if (lista==NULL)
```

```
    {
```

```
        novi->sledeci=novi;
```

```
        novi->prethodni=novi;
```

```
        return novi;
```

```
    }
```

```
    else
```

```
    {
```

```
        novi->prethodni=lista;
```

```
        novi->sledeci=lista->sledeci;
```

```
        lista->sledeci->prethodni=novi;
```

```
        lista->sledeci=novi;
```

```
        return novi;
```

```
    }
```

```
}
```

```
/* Ispis liste */
```

```
void ispisi(cvor* lista)
```

```
{
```

```
    if (lista!=NULL)
```

```
    {    cvor* tekuci=lista;
```

```
        do
```

```
        {    printf("%d\n",tekuci->broj);
```

```
            tekuci=tekuci->sledeci;
```

```
        } while (tekuci!=lista);
```

```
    }
```

```
}
```

```
/* Izbacivanje datog cvora iz liste */
```

```
cvor* izbaci(cvor* lista)
{
    if (lista!=NULL)
    {
        cvor* sledeci=lista->sledeci;
        if (lista==lista->sledeci)
        {
            printf("Pobednik %d\n",lista->broj);
            free(lista);
            return NULL;
        }

        printf("Ispada %d\n",lista->broj);

        lista->sledeci->prethodni=lista->prethodni;
        lista->prethodni->sledeci=lista->sledeci;
        free(lista);
        return sledeci;
    }
    else return NULL;
}

main()
{
    /* Umecemo petoro dece u listu */
    cvor* lista = NULL;
    lista=ubaci(1,lista);
    lista=ubaci(2,lista);
    lista=ubaci(3,lista);
    lista=ubaci(4,lista);
    lista=ubaci(5,lista);
    lista=lista->sledeci;

    int smer = 0;
    /* Dok ima dece u listi */
    while(lista!=NULL)
    {
        int i;

        /* brojimo 13 slogova u krug i
           u svakom brojanju menjamo smer obilaska*/
        for (i=1; i<=13; i++)
            lista = 1-smer ? lista->sledeci : lista->prethodni;

        lista=izbaci(lista);
        smer = smer ? 0 : 1;
    }
    ispisi(lista);
}
```

2.2 Zadaci za vežbu

Zadatak 1 Brojeve sa ulaza smeštati u listu sve dok se ne unese nula, a zatim dobijenu listu ispisati na izlaz.

1. Zadatak realizovati dodavanjem elemenata liste na početak liste.
2. Zadatak realizovati tako da listu koja se formira bude sortirana.
3. Zadatak realizovati dodavanjem elemenata liste na kraj liste a listu ispisati unazad.

Zadatak 2 Grupa od n plesača (na čijim kostimima su u smeru kazaljke na satu redom brojevi od 1 do n) izvodi svoju plesnu tačku tako što formiraju krug iz kog najpre izlazi k -ti plesač (odbrojava se počev od plesača označenog brojem 1 u smeru kretanja kazaljke na satu). Preostali plesači obrazuju manji krug iz kog opet izlazi k -ti plesač (odbrojava se počev od sledećeg suseda prethodno izbačenog, opet u smeru kazaljke na satu). Izlasci iz kruga se nastavljaju sve dok svi plesači ne budu isključeni. Celi brojevi n, k ($k < n$) se učitavaju sa standardnog ulaza. Napisati program koji će na standardni izlaz ispisati redne brojeve plesača u redosledu napuštanja kruga.

PRIMER: za $n = 5, k = 3$ redosled izlaska je 3 1 5 2 4.