Programiranje II

Beleške sa vežbi

Smer *Informatika*

Matematički fakultet, Beograd

Sana Stojanović

12.06.07.

Sadržaj

1 Rekurzije sa stablima

1.1 Ispis stabla po nivoima

- Napisati program koji čvorove binarnog stabla štampa u redosledu koji odgovara nivoima (prvo element koji se nalazi u prvom nivou, pa elemente drugog nivoa i tako redom). Za ispis elemenata stabla po nivoima koristićemo red.
- Opisati strukturu reda koji će se koristiti za čuvanje cvorova stabla kao i osnovne funkcije za rad sa tim redom.

```
/* Struktura reda u kojoj cemo cuvati pokazivace na elemente stabla */
typedef struct red
{
   cvor *el;
   struct red * sl;
} red;

/* Funkcija koja pravi jedan element reda. Njen argument je pokazivac
   na cvor stabla koji ubacujemo */
red * napravi_red(cvor *el)
{
   red * novi = (red *)malloc(sizeof(red));

   if (novi==NULL)
   {
      fprintf(stderr, "greska\n");
      exit(1);
   }

   novi->el = el;
   novi->sl = NULL;

   return novi;
}

/* Ubacivanje elementa u red, element ide na kraj reda i menja se kraj reda */
void ubaci_u_red(red ** pred, red ** pkraj, cvor *el)
{
   red * novi = napravi_red(el);

   /* Ako je red bio prazan dodajemo element i upucujemo oba
      pokazivaca na njega */
   if (*pkraj == NULL)
   {
```

```
      *pkraj = novi;
      *pred = novi;
   }
   else
   {
      /* U suprotnom samo nadovezujemo tekuci element na kraj reda.
         Pocetak reda ostaje nepromenjen. */
      (*pkraj)->sl = novi;
      *pkraj = novi;
   }
}

/* Izbacivanje prvog elementa iz reda. Menja se pocetak reda. */
void izbaci_iz_reda(red **poc, red **kraj)
{
   red *pom;
   pom = *poc;
   if (*poc !=NULL)
   {
      *poc = (*poc)->sl;
      free(pom);
      if(*poc == NULL) *kraj = NULL;
   }
}
```

- Napisati main-funkciju koja ispisuje elemente drveta u redosledu koji odgovara nivoima.

```
int main()
{
    cvor *drvo = NULL;
    int i;
    red *poc = NULL, *kraj=NULL; /* Pokazivaci na pocetak, odnosno
                                    na kraj reda */

    /* Petlja u kojoj popunjavamo elemente stabla */
    while(1)
    {
       printf("Unesi sledeci cvor stabla, 0 za kraj:\n");
       scanf("%d", &i);
       if (i==0) break;
       ubaci_u_drvo(&drvo, i);
    }
```

```
    /* U red ubacujemo koren stabla */
    ubaci_u_red(&poc, &kraj, drvo);

    /* Dok god ima elemenata u redu dodajemo njihove potomke. */
    while(poc)
    {
        /* Stampamo prvi element reda... */
        printf("%d ", poc->el->br);

        /* ...ako postoje, dodajemo njegove potomke na kraj reda... */
       if ((poc)->el->l)
            ubaci_u_red(&poc, &kraj, (poc)->el->l);
       if ((poc)->el->d)
            ubaci_u_red(&poc, &kraj, (poc)->el->d);

       /* ...i brisemo ga iz reda. */
       izbaci_iz_reda(&poc,&kraj);
    }

    obrisi_drvo(drvo);
}
```

1.2 Razni zadaci sa stablima

1. Napisati program koji sa standardnog ulaza čita tekst i ispisuje broj pojavljivanja svake od reči koje su se javile u tekstu. Čuvanje svih reči i broj pojavljivanja realizovati preko uređenog binarnog stabla.

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

/* Cvor drveta sadrzi ime reci i broj njenih pojavljivanja */
typedef struct _cvor {
     char ime[80];
     int br_pojavljivanja;
     struct _cvor* levo, *desno;
} cvor;


/* Funkcija ispisuje drvo u inorder redosledu */
void ispisi_drvo(cvor* drvo)
{
    if (drvo!=NULL)
     {
        ispisi_drvo(drvo->levo);
```

```
        printf("%s %d\n",drvo->ime,drvo->br_pojavljivanja);

        ispisi_drvo(drvo->desno);
     }
}


/* Funkcija uklanja binarno drvo iz memorije */
void obrisi_drvo(cvor* drvo)
{
    if (drvo!=NULL)
     {
        obrisi_drvo(drvo->levo);

        obrisi_drvo(drvo->desno);

        free(drvo);
     }
}

/* Funkcija ubacuje datu rec u dato drvo i vraca pokazivac na
   koren drveta */
cvor* ubaci(cvor* drvo, char rec[])
{
    /* Ukoliko je drvo prazno gradimo novi cvor */
    if (drvo==NULL)
    {
        cvor* novi_cvor=(cvor*)malloc(sizeof(cvor));
        if (novi_cvor==NULL)
        {
           printf("Greska prilikom alokacije memorije\n");
           exit(1);
        }

        /* Ubacujemo rec u koren drveta i postavljamo broj pojavljivanja na 1 */
        strcpy(novi_cvor->ime, rec);
        novi_cvor->br_pojavljivanja=1;

        return novi_cvor;
    }

    /* Ako drvo nije prazno proveravamo da li se nova rec nalazi u korenu
       drveta */
    int cmp = strcmp(rec, drvo->ime);

    /* Ukoliko se rec nalazi u korenu drveta uvecavamo njen broj
```

```
       pojavljivanja za 1 */
    if (cmp==0)
    {
        drvo->br_pojavljivanja++;
        return drvo;
    }

    /* Ukoliko je rec koju ubacujemo leksikografski ispred reci koja je u
       korenu drveta, rec ubacujemo u levo podstablo */
    if (cmp<0)
    {
        drvo->levo=ubaci(drvo->levo, rec);
        return drvo;
    }

    /* Ukoliko je rec koju ubacujemo leksikografski iza reci koja je u
       korenu drveta, rec ubacujemo u desno podstablo */
    if (cmp>0)
    {
        drvo->desno=ubaci(drvo->desno, rec);
        return drvo;
    }
}

/* Pomocna funkcija koja cita rec sa standardnog ulaza i vraca njenu
   duzinu, odnosno -1 ukoliko se naidje na EOF */
int getword(char word[], int lim)
{
    int c, i=0;

    while (!isalpha(c=getchar()) && c!=EOF)
             ;

    if (c==EOF)
          return -1;
    do
    {
          word[i++]=c;
    }while (i<lim-1 && isalpha(c=getchar()));

    word[i]='\0';
    return i;
}

main()
{
```

```
   /* Drvo je na pocetku prazno */
   cvor* drvo=NULL;
   char procitana_rec[80];

   /* Citamo rec po rec dok ne naidjemo na kraj datoteke i
      ubacujemo ih u drvo */
   while(getword(procitana_rec,80)!=-1)
      drvo=ubaci(drvo,procitana_rec);

   /* Ispisujemo drvo */
   ispisi_drvo(drvo);

   /* Uklanjamo ga iz memorije */
   obrisi_drvo(drvo);
}
```

2. Napisati program koji broji pojavljivanja svih etiketa u HTML datoteci - etikete ispisati opadajuce po broju pojavljivanja.

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

/* Struktura cvora drveta u kome se cuvaju etikete zajedno sa brojem
   pojavljivanja. Drvo je pretrazivacko i sortirano je lekikografski po
   etiketama */
typedef struct _node{
    char tag[30];
    int num;
    struct _node *l,*r;
} node;


/* Zbog sortiranja po broju cvorova, paralelno sa strukturom
   drveta, odrzavamo niz pokazivaca na njegove cvorove */
node* nodes[100];

/* Dosadasnji broj cvorova drveta (razlicitih etiketa) */
int num_nodes = 0;


/* Funkcija kreira cvor koji sadrzi datu etiketu */
node* make_node(char *tag)
{
    node* new_node = (node*) malloc(sizeof(node));
```

```
    if(new_node == NULL)
    {
        fprintf(stderr,"Greska prilikom alokacije memorije\n");
        exit(1);
    }

    strcpy(new_node->tag, tag);
    new_node->l=NULL;
    new_node->r=NULL;
    new_node->num = 1;

    /* Dopisujemo cvor u niz postojecih cvorova */
    nodes[num_nodes++] = new_node;

    return new_node;
}

/* Funkcija umece datu etiketu u postojece drvo. Ukoliko etiketa
   postoji, povecava se njen broj pojavljivanja */
void insert(node** ptree, char tag[])
{
    int cmp;
    if(*ptree==NULL)
    {
        *ptree = make_node(tag);
        return;
    }

    cmp = strcmp(tag, (*ptree)->tag);

    if (cmp < 0)
        insert(&((*ptree)->l), tag);
    else if (cmp > 0)
        insert(&((*ptree)->r), tag);
    else
        (*ptree)->num++;
}

/* Funcija za ispis drveta */
void print(node* tree)
{
    if(tree != NULL)
    {
        print(tree->l);
        printf("%-10s - %3d\n", tree->tag, tree->num);
        print(tree->r);
```

```
    }
}

/* Funkcija koja uklanja drvo */
void remove_tree(node* tree)
{
    if(tree != NULL)
    {
        remove_tree(tree->l);
        remove_tree(tree->r);
        free(tree);
    }
}


/* Funkcija poredjenja za poziv ugradjene funkcije qsort. Porede se
   dva cvora drveta na osnovu broja pojavljivanja etiketa */
int compare(const void* pa, const void* pb)
{
    return (*((node**)pb))->num - (*((node**)pa))->num;
}

/* Funkcija ucitava etiketu iz date datoteke. Funkcija vraca
   logicku vrednost koja indikuje da li je etiketa uspesno
   procitana */
int get_tag(FILE* f, char tag[])
{
    int c;
    int i = 0;

    /* Preskacemo sve do prvog znaka '<' ili kraja */
    while ((c = fgetc(f)) != EOF && c != '<')
        ;

    /* Nije bilo vise etiketa */
    if (c == EOF)
        return 0;

    /* Citamo prvi karakter etiketa*/
    c = fgetc(f);

    /* Gutamo / kod zatvorenih etiketa */
    if (c == '/')
        c = fgetc(f);

    /* Ime etikete cine slova */
```

```
    while(isalpha(c))
    {
        tag[i++] = c;
        c = fgetc(f);
    }
    tag[i] = '\0';

    /* Preskacemo sve do > ili do kraja */
    while (c != '>' && c != EOF)
        c = fgetc(f);

    if (c == EOF)
        return 0;

    return 1;
}

main(int argc, char* argv[])
{
    /* Tekuca procitana etiketa */
    char tag[30];

    /* Drvo koje je leksikografski sortirano zbog brze pretrage */
    node* tree = NULL;

    /* Datoteka iz koje se cita */
    FILE* f;
    int i;

    /* Proverava se korektnost argumenata komandne linije */
    if (argc < 2)
    {
        printf("Upotreba : %s ime_datoteke\n", argv[0]);
        exit(1);
    }

    /* Otvara se datoteka */
    f = fopen(argv[1], "r");
    if (f == NULL)
    {
        fprintf(stderr, "Greska prilikom otvaranja %s\n", argv[1]);
        return;
    }

    /* Kreiramo drvo na osnovu sadrzaja datoteke */
    while(get_tag(f, tag) == 1)
```

```
        insert(&tree, tag);

    /* Zatvaramo datoteku */
    fclose(f);

    /* Sortiramo cvorove niza na osnovu broja pojavljivanja etiketa */
    qsort(nodes, num_nodes, sizeof(node*), &compare);

    /* Ispisujemo sortirane cvorove */
    for (i = 0; i<num_nodes; i++)
        printf("%-10s - %3d\n", nodes[i]->tag, nodes[i]->num);

    /* Uklanjamo drvo */
    remove_tree(tree);

}
```

2 Primeri korišćenja funkcija *fseek* i *ftell*

1. Program koji računa dužinu fajla u bajtovima.

```
#include <stdio.h>

long filesize(FILE *f);

int main(void)
{
   FILE *f;

   f = fopen("datoteka.txt", "w+"); //otvaranje fajla
   fprintf(f, "Ovo je test.");
   printf("Velicina fajla datoteka.txt je %ld bajtova\n", filesize(f));
   fclose(f);
   return 0;
}

//Funkcija koja racuna duzinu fajla
long filesize(FILE *f)
{
   long trenutna_pozicija, duzina;  //trenutna pozicija i duzina fajla

   trenutna_pozicija = ftell(f);    //pamti se trenutna pozicija
                                    //da bismo mogli na nju da se vratimo

   fseek(f, 0, SEEK_END);           //pozicioniranje na kraj fajla
```

```
    duzina = ftell(f); //uzimamo poziciju u broju bajtova od pocetka
                       //fajla, to je duzina

    fseek(f, trenutna_pozicija, SEEK_SET); //vracamo se nazad na zapamcenu
                                           //poziciju
    return duzina;

}
```

2. Napisati program koji u datoteci koja se zadaje kao prvi argument komandne linije zamenjuje sva pojavljivanja niske koja se zadaje kao drugi argument komandne linije niskom koja se zadaje kao treći argument komandne linije.

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

int main(int argc, char* argv[])
{
   FILE *f;
   int c;
   int i;
   long pos;
   int duzina;


   f = fopen(argv[1], "r+");

   while ((c = fgetc(f))!=EOF)
   {
        if (c == argv[2][0])//ako smo naisli na prvi karakter niske
        {
            pos = ftell(f); //zapamtimo trenutnu poziciju

            duzina = strlen(argv[2]);

            //petlja koja ako prepozna celu rec vrati se na pocetak i
            //prepise je
            for(i=1; i<duzina; i++)
                //ako se karakteri ne poklapaju izlazimo iz petlje
                if (argv[2][i]!=fgetc(f))
                    break;

            //ako je bila pronadjena cela rec prepisujemo je trecim argumentom
            //komandne linije
```

```
            if (i==duzina)
            {

                //vracamo se na pocetak reci da bismo je prepisali
                fseek(f,pos-1,SEEK_SET);

                for(i=0; argv[3][i]; i++)
                    fputc(argv[3][i], f);

                fseek(f,0,SEEK_CUR); //pozicionira na tu poziciju
                                     //(jer inace fputs smatra da upisuje
                                     //na kraj fajla pa fseek pocisti taj
                                     //fleg
            }
            else
                //vracamo se na pocetak reci ali iza prvog slova
                fseek(f, pos, SEEK_SET);
        }
    }
    fclose(f);
}
```