Programiranje 2

Beleške sa vežbi

Školska 2006/2007 godina

Matematički fakultet, Beograd

Jelena Tomašević

May 12, 2007

Sadržaj

1

Programski jezik C

[1]

1.1 Stek

Primer 1 *Provera uparenosti HTML etiketa - stek se implementira preko liste.*

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

/* Maksimalna duzina etikete */
#define MAX_TAG 100

#define OPEN   1
#define CLOSED 2
#define ERROR 0

/* Funkcija ucitava sledecu etiketu i smesta njen naziv u niz s duzine max.
   Vraca OPEN za otvorenu etiketu, CLOSED za zatvorenu etiketu,
   odnosno ERROR inace */
int gettag(char s[], int max)
{   int c, i;
    int type=OPEN;

    /* Preskacemo sve do znaka '<' */
    while ((c=getchar())!=EOF && c!='<')
                    ;
    /* Nismo naisli na etiketu */
    if (c==EOF)
           return ERROR;

    /* Proveravamo da li je etiketa zatvorena */
    if ((c=getchar())=='/')
       type = CLOSED;
```

[1] Zasnovano na primerima sa sajta `http://www.matf.bg.ac.yu/~filip`

```
    else
       ungetc(c,stdin);

    /* Citamo etiketu dok nailaze slova i smestamo ih u nisku*/
    for (i=0; isalpha(c=getchar()) && i<max-1; s[i++] = c)
                    ;
    s[i]='\0';

    /* Preskacemo atribute do znaka > */
    while (c!=EOF && c!='>')
    c = getchar();

    /* Greska ukoliko nismo naisli na '>' */
    return c=='>' ? type : ERROR;
}

/*****************************************************************/
/* Stek ce biti implementiran koriscenjem liste */
typedef struct node
{
    char string[MAX_TAG];
    struct node* next;
} NODE;

/* Funkcija postavlja dati string na stek */
void push(NODE** pstack, char* s)
{
    NODE* tmp = (NODE*)malloc(sizeof(NODE));
    if (tmp == NULL)
    {
        fprintf(stderr, "Greska prilikom alokacije memorije\n");
        exit(1);
    }
    strcpy(tmp->string, s);
    tmp->next = *pstack;
    *pstack = tmp;
}

/* Funkcija cita podatak sa vrha steka */
char* peek(NODE* stack)
{
    /* Stek ne sme da bude prazan */
    assert(stack != NULL);

    return stack->string;
}

/* Funkcija uklanja podatak sa vrha steka */
void pop(NODE** pstack)
{
    /* Ukoliko je stek prazan ne radimo nista */
    if (*pstack == NULL)
```

```
        return;

    NODE* tmp = (*pstack)->next;
    free(*pstack);
    *pstack = tmp;
}

/* Funkcija proverava da li je dati stek prazan */
int empty(NODE* stack)
{
    return stack == NULL;
}
/* ********************************************************************** */


main()
{
    char tag[MAX_TAG];
    NODE* stack = NULL;

    int type;
    while ((type = gettag(tag,MAX_TAG)) != ERROR)
    {
        if (type == OPEN)
        {
            /* Svaku otvorenu etiketu stavljamo na stek */
            push(&stack, tag);
            printf("Postavio <%s> na stek\n", peek(stack));
        }
        else
        {
            /* Za zatvorene etikete proveravamo da li je stek prazan
            odnosno da li se na vrhu steka nalazi odgovarajuca otvorena etiketa */
            if (!empty(stack) && strcmp(peek(stack), tag) == 0)
            {
                printf("Skidam <%s> sa steka\n", peek(stack));
                /* Uklanjamo etiketu sa steka */
                pop(&stack);
            }
            else
            {
                /* Prijavljujemo gresku */
                printf("Neodgovarajuce : </%s>\n",tag);
                exit(1);
            }
        }
    }

    /* Proveravamo da li je stack ispraznjen */
    if (!empty(stack))
        fprintf(stderr, "Nisu sve etikete zatvorene\n");
}
```

1.2 Drveta

1.2.1 Binarno pretraživačko drvo

Primer 2 *Binarno pretraživačko drvo - drvo sadrži cele brojeve. Ubacivanje realizovano preko ***

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

/* Struktura jednog cvora drveta */
typedef struct _cvor
{
    /* Podatak */
    int broj;
    /* Pokazivac na levo i desno podstablo */
    struct _cvor *l, *d;
} cvor;

/* Pomocna funkcija koja kreira novi cvor na osnovu datog broja */
cvor* napravi_cvor(int b)
{
    cvor* novi = (cvor*)malloc(sizeof(cvor));
    if (novi == NULL)
    {
        fprintf(stderr, "Greska prilikom alokacije memorije");
        exit(1);
    }

    /* Postavljamo brojnu vrednost */
    novi->broj = b;

    /* Novi cvor se kreira kao list */
    novi->l = NULL;
    novi->d = NULL;
    return novi;
}

/* Rekurzivna funkcija koja ubacuje dati broj u dato drvo */
void ubaci_u_drvo(cvor** pdrvo, int b)
{
    /* Ukoliko je drvo prazno kreira se novi cvor */
    if (*pdrvo == NULL)
    {
        *pdrvo = napravi_cvor(b);
        return;
    }

    /* Ukoliko je broj koji se ubacuje manji od broja u korenu,
       rekurzivno ga ubacujemo u levo podstablo.
       Ukoliko je broj koji se ubacuje veci od broja u korenu,
       rekurzivno ga ubacujemo u desno podstablo.
    */
```

```
    if (b < (*pdrvo)->broj)
        ubaci_u_drvo(&((*pdrvo)->l), b);
    else if (b > (*pdrvo)->broj)
        ubaci_u_drvo(&((*pdrvo)->d), b);
}


/* Rekurzivna funkcija koja proverava da li dati broj postoji u drvetu */
int pronadji(cvor* drvo, int b)
{
    /* U praznom drvetu ne postoji broj */
    if (drvo == NULL)
        return 0;

    /* Ukoliko je jednak vrednosti u korenu, onda postoji */
    if (drvo->broj == b)
        return 1;

    /* Ukoliko je broj koji trazimo manji od vrednosti u korenu,
       trazimo ga samo u levom podstablu, a inace ga trazimo
       samo u desnom podstablu */
    if (b < drvo->broj)
        return pronadji(drvo->l, b);
    else
        return pronadji(drvo->d, b);
}

/* Rekurzivna funkcija koja ispisuje drvo u inorder redosledu */
void ispisi_drvo(cvor* drvo)
{
    if (drvo != NULL)
    {
        ispisi_drvo(drvo->l);
        printf("%d ", drvo->broj);
        ispisi_drvo(drvo->d);
    }
}


/* Rekurzivna funkcija koja uklanja drvo. Obilazak mora biti postorder. */
void obrisi_drvo(cvor* drvo)
{
    if (drvo != NULL)
    {
        obrisi_drvo(drvo->l);
        obrisi_drvo(drvo->d);
        free(drvo);
    }
}

main()
{
```

```
    cvor* drvo = NULL;
    ubaci_u_drvo(&drvo, 1);
    ubaci_u_drvo(&drvo, 8);
    ubaci_u_drvo(&drvo, 3);
    ubaci_u_drvo(&drvo, 5);
    ubaci_u_drvo(&drvo, 7);
    ubaci_u_drvo(&drvo, 6);
    ubaci_u_drvo(&drvo, 9);

    if (pronadji(drvo, 3))
        printf("Pronadjeno 3\n");
    if (pronadji(drvo, 2))
        printf("Pronadjeno 2\n");
    if (pronadji(drvo, 7))
        printf("Pronadjeno 7\n");

    ispisi_drvo(drvo);
    putchar('\n');

    obrisi_drvo(drvo);
}
```

Primer 3 *Binarno pretraživacko drvo - drvo sadrži cele brojeve. Ubacivanje realizovano preko eksplicitnog vraćanja korena rezultujućeg drveta.*

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

typedef struct _cvor
{
    int broj;
    struct _cvor *l, *d;
} cvor;

cvor* napravi_cvor(int b)
{
    cvor* novi = (cvor*)malloc(sizeof(cvor));
    if (novi == NULL)
    {
        fprintf(stderr, "Greska prilikom alokacije memorije");
        exit(1);
    }
    novi->broj = b;
    novi->l = NULL;
    novi->d = NULL;
    return novi;
}

cvor* ubaci_u_drvo(cvor* drvo, int b)
{
    if (drvo == NULL)
        return napravi_cvor(b);
```

```
    if (b < drvo->broj)
        drvo->l = ubaci_u_drvo(drvo->l, b);
    else
        drvo->d = ubaci_u_drvo(drvo->d, b);

    return drvo;
}

/* Funkcija proverava da li dati broj postoji u drvetu */
int pronadji(cvor* drvo, int b)
{
    if (drvo == NULL)
        return 0;

    if (drvo->broj == b)
        return 1;

    if (b < drvo->broj)
        return pronadji(drvo->l, b);
    else
        return pronadji(drvo->d, b);
}

void ispisi_drvo(cvor* drvo)
{
    if (drvo != NULL)
    {
        ispisi_drvo(drvo->l);
        printf("%d ", drvo->broj);
        ispisi_drvo(drvo->d);
    }
}


void obrisi_drvo(cvor* drvo)
{
    if (drvo != NULL)
    {
        obrisi_drvo(drvo->l);
        obrisi_drvo(drvo->d);
        free(drvo);
    }
}

main()
{
    cvor* drvo = NULL;
    drvo = ubaci_u_drvo(drvo, 1);
    drvo = ubaci_u_drvo(drvo, 8);
    drvo = ubaci_u_drvo(drvo, 5);
    drvo = ubaci_u_drvo(drvo, 3);
    drvo = ubaci_u_drvo(drvo, 7);
```

```
    drvo = ubaci_u_drvo(drvo, 6);
    drvo = ubaci_u_drvo(drvo, 9);

    if (pronadji(drvo, 3))
        printf("Pronadjeno 3\n");
    if (pronadji(drvo, 2))
        printf("Pronadjeno 2\n");
    if (pronadji(drvo, 7))
        printf("Pronadjeno 7\n");

    ispisi_drvo(drvo);
    putchar('\n');

    obrisi_drvo(drvo);
}
```

Primer 4 *Rekurzivne funkcije za rad sa celobrojnim stablima (ne obavezno pretrazivackim): broj_cvorova, broj_listova, suma_cvorova, dubina, najveci_cvor, ...*

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

typedef struct _cvor
{
    int broj;
    struct _cvor *l, *d;
} cvor;

cvor* napravi_cvor(int b)
{
    cvor* novi = (cvor*)malloc(sizeof(cvor));
    if (novi == NULL)
    {
        fprintf(stderr, "Greska prilikom alokacije memorije");
        exit(1);
    }
    novi->broj = b;
    novi->l = NULL;
    novi->d = NULL;
    return novi;
}

void ubaci_u_drvo(cvor** drvo, int b)
{
    if (*drvo == NULL)
    {
        *drvo = napravi_cvor(b);
        return;
    }

    if (b < (*drvo)->broj)
        ubaci_u_drvo(&((*drvo)->l), b);
    else
```

```
        ubaci_u_drvo(&((*drvo)->d), b);
}

void ispisi_drvo(cvor* drvo)
{
    if (drvo != NULL)
    {
        ispisi_drvo(drvo->l);
        printf("%d ", drvo->broj);
        ispisi_drvo(drvo->d);
    }
}


void obrisi_drvo(cvor* drvo)
{
    if (drvo != NULL)
    {
        obrisi_drvo(drvo->l);
        obrisi_drvo(drvo->d);
        free(drvo);
    }
}

/* Izracunava sumu svih elemenata u cvorovima drveta */
int suma_cvorova(cvor* drvo)
{
    if (drvo == NULL)
        return 0;
    return  suma_cvorova(drvo->l) +
        drvo->broj +
        suma_cvorova(drvo->d);
}

/* Izracunava broj cvorova datog drveta */
int broj_cvorova(cvor* drvo)
{
    if (drvo == NULL)
        return 0;
    return  broj_cvorova(drvo->l) +
        1 +
        broj_cvorova(drvo->d);
}

/* Izracunava broj listova datog drveta */
int broj_listova(cvor* drvo)
{
    if (drvo == NULL)
        return 0;

    /* Cvor je list ukoliko nema ni jednog naslednika */
    if (drvo->l == NULL && drvo->d == NULL)
```

```
        return 1;

    return  broj_listova(drvo->l) +
        broj_listova(drvo->d);
}

/* Izracunava sumu svih elemenata u listovima drveta */
int suma_listova(cvor* drvo)
{
    if (drvo == NULL)
        return 0;

    if (drvo->l == NULL && drvo->d == NULL)
        return drvo->broj;

    return  suma_listova(drvo->l) +
        suma_listova(drvo->d);
}

/* Ispisuje sve elemente u listovima drveta */
void ispisi_listove(cvor* drvo)
{
    if (drvo == NULL)
        return;

    ispisi_listove(drvo->l);

    if (drvo->l == NULL && drvo->d == NULL)
        printf("%d ", drvo->broj);

    ispisi_listove(drvo->d);
}

/* Izracunava vrednost najveceg cvora u proizvoljnom drvetu.
   Vraca -1 ukoliko je drvo prazno */
int najveci_cvor(cvor* drvo)
{

    if (drvo == NULL)
        return -1;
    else
    {
        int max_l = najveci_cvor(drvo->l);
        int max_d = najveci_cvor(drvo->d);

        return max_l<max_d ?
            (max_d<drvo->broj?drvo->broj:max_d) :
            (max_l<drvo->broj?drvo->broj:max_l);
    }
}

/* Izracunava dubinu (broj nivoa drveta) */
```

```
int dubina(cvor* drvo)
{
    if (drvo == NULL)
        return 0;
    else
    {
        int dl = dubina(drvo->l);
        int dd = dubina(drvo->d);

        return dl<dd ? dd + 1 : dl + 1;
    }
}

main()
{
    cvor* drvo = NULL;
    ubaci_u_drvo(&drvo, 1);
    ubaci_u_drvo(&drvo, 8);
    ubaci_u_drvo(&drvo, 5);
    ubaci_u_drvo(&drvo, 3);
    ubaci_u_drvo(&drvo, 7);
    ubaci_u_drvo(&drvo, 6);
    ubaci_u_drvo(&drvo, 9);

    printf("Suma cvorova : %d\n", suma_cvorova(drvo));
    printf("Broj cvorova : %d\n", broj_cvorova(drvo));
    printf("Broj listova : %d\n", broj_listova(drvo));
    printf("Suma listova : %d\n", suma_listova(drvo));
    printf("Najveci cvor : %d\n", najveci_cvor(drvo));
    printf("Dubina : %d\n", dubina(drvo));
    printf("Listovi : ");
    ispisi_listove(drvo);
    putchar('\n');

    obrisi_drvo(drvo);
}
```

Primer 5 *Program sa ulaza cita tekst i ispisuje broj pojavljivanja svake od reci koje su se javljale u tekstu. Verzija sa binarnim pretrazivackim drvetom.*

```
#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);
        }
        strcpy(novi_cvor->ime, rec);
        novi_cvor->br_pojavljivanja=1;
        return novi_cvor;
    }
    int cmp = strcmp(rec, drvo->ime);

    /* Ukoliko rec vec postoji u drvetu uvecavamo njen broj pojavljivanja */
    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);

}
```

Primer 6 *Program koji broji pojavljivanja svih etiketa u HTML datoteci - etikete se ispisuju opadajuci 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);

}
```

1.3 Zadaci za vežbu

Zadatak 1 ***September, 2005.*** *Napisati program koji na standardni izlaz ispisuje naziv (BEZ ATRIBUTA) najčešće korišćene etikete u datoteci ulaz.htm. Ako ima više takvih, ispisati ma koju. Koristiti uređeno binarno stablo. Pretpostaviti da je ulazna datoteka sintaksno korektna.*

Zadatak 2 ***Drugi kolokvijum za II tok 2004.godine - rad na računaru*** *Napisati program koji iz tekstualne datoteke čiji je put dat u argumentu komandne linije učitava različite prirodne brojeve i:*

1. *dodaje ih redom u uredjeno binarno stablo*
2. *u dobijenom drvetu izračunava dužinu najdužeg puta od korena do nekog lista i*
3. *štampa u rastućem poretku (bez ponavljanja) sve brojeve koji su nalaze na putevima te dužine od korena do listova.*