Programiranje II

Beleške sa vežbi

Smer *Informatika*

Matematički fakultet, Beograd

Sana Stojanović

20.03.08.

Sadržaj

1 Dinamička alokacija, funkcija *realloc* - ponavljanje

1. Napisati program koji sa standardnog ulaza unosi cele brojeve dok se ne unese -1 kao oznaka za kraj unosa. Broj celih brojeva nije unapred poznat. Zadatak resiti korišćenjem dinamičke alokacije.

```
/* Program demonstrira niz kome se velicina tokom rada povecava
   - verzija sa funkcijom realloc */

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

/* Konstanta koja nam odredjuje za koliko elemenata cemo prosirivati
   niz prilikom jednog poziva funkcije realloc */
#define KORAK 10

int main()
{
    int* a = NULL;      /* Niz je u pocetku prazan i postavljamo pokazivac
                           na NULL da bi mogao biti prosledjen funkciji
                           realloc */
    int* pom;           /* Pomocni pokazivac koji ce nam sluziti da
                           preuzme povratnu vrednost funkcije realloc
                           pre nego budemo mogli da je dodelimo pokazivacu
                           a */

    int duzina = 0, alocirano = 0;  /* Duzina predstavlja broj popunjenih
                                       elemenata niza, dok alocirano
                                       predstavlja broj alociranih
                                       elemenata */

    int n, i;

    do
    {
        printf("Unesi ceo broj (-1 za kraj): ");
        scanf("%d", &n);

        /* Ukoliko nema vise slobodnih mesta, vrsi se prosirivanje */
        if (duzina == alocirano)
        {
            /* Niz se prosiruje za 10 elemenata vise */
            alocirano = alocirano + KORAK;

            /* Poziv funkcije realloc za a = NULL, ekvivalentan je pozivu
```

```
                funkcije malloc(alocirano*sizeof(int)); */
            pom = realloc(a, alocirano*sizeof(int));

            /* Ukoliko funkcija realloc vrati pokazivac koji se
               razlikuje od NULL realokacija je uspela i dodeljujemo
               vrednost pomocnog pokazivaca pokazivacu a */
            if (pom != NULL)
                a = pom;
            else
                {
                    /* Ako realloc vrati NULL (a kao drugi parametar joj
                       nije prosledjena duzina 0) to znaci da realokacija
                       nije uspela i prostor na koji pokazuje pokazivac
                       a nece biti oslobodjen pa to moramo uraditi rucno */
                    free(a);
                    exit 1;
                }
        }
        /* Sada kada sigurno ima mesta u nizu upisujemo procitanu
           vrednost na tekucu poziciju u nizu i uvecavamo duzinu niza
           za 1 */
        a[duzina++] = n;

    } while (n != -1);

    /* Ispisujemo niz a */
    printf("Uneto je %d brojeva. Alocirano je mesta za %d brojeva\n",
        duzina, alocirano);

    printf("Brojevi su : ");
    for (i = 0; i<duzina; i++)
        printf("%d ", a[i]);

    /* Oslobadjamo niz a */
    free(a);

    return 0;
}
```

2. Napisati program koji sa standardnog ulaza unosi cele brojeve dok se ne unese -1 kao oznaka za kraj unosa. Broj celih brojeva nije unapred poznat. Zadatak resiti korišćenjem dinamičke alokacije bez funkcije *realloc*.

```
/* Program demonstrira niz kome se velicina tokom rada povecava
   - verzija sa rucnom realokacijom */

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

/* Niz ce se uvecavati u koracima od po 10 brojeva */
#define KORAK 10

int main()
{
    /* Niz je u pocetku prazan */
    int* a = NULL;

    /* Duzina predstavlja broj popunjenih elemenata,
       dok alocirano predstavlja broj alociranih elemenata */
    int duzina = 0, alocirano = 0;
    int n, i;


    do
    {
        printf("Unesi ceo broj (-1 za kraj): ");
        scanf("%d", &n);

        /* Ukoliko nema vise slobodnih mesta, vrsi se prosirivanje */
        if (duzina == alocirano)
        {
            /* Kreira se novi niz */
            int* new_a;

            /* Za njega se alocira 10 elemenata vise od prethodnog */
            alocirano = alocirano + KORAK;
            new_a = malloc(alocirano*sizeof(int));

            if (new_a == NULL)
                {
                    free(a);
                    exit 1;
                }

            /* Kopira se sadrzaj starog niza u novi */
            for (i = 0; i<duzina; i++)
                new_a[i] = a[i];

            /* Oslobadja se stari niz */
```

```
            free(a);

            /* Stari niz postaje novi */
            a = new_a;
        }

        /* Ucitani broj se upisuje u niz */
        a[duzina++] = n;

    } while (n != -1);

    /* Ispisujemo niz a */
   printf("Uneto je %d brojeva. Alocirano je mesta za %d brojeva\n",
        duzina, alocirano);
    printf("Brojevi su : ");
    for (i = 0; i<duzina; i++)
        printf("%d ", a[i]);

    /* Oslobadjamo niz a */
    free(a);

    return 0;
}
```

2 Dinamička alokacija, funkcija *realloc* - razni zadaci

1. Napisati program koji za unetu granicu g sa standardnog ulaza određuje niz Fibonačijevih brojeva do prvog Fibonačijevog broja koji je veći od g (uključujući i taj broj). Fibonačijevi brojevi se definišu na sledeći način:

   $f_0 = 0$, $f_1 = 1$

   $f_i = f_{i-1} + f_{i-2}$, za $i > 1$

   Zadatak rešiti korišćenjem dinamičke alokacije.

   Primer:

   ```
   Za uneto g = 10 dobijamo niz: 0, 1, 1, 2, 3, 5, 8, 13
   Za uneto g = 4 dobijamo niz: 0, 1, 1, 2, 3, 5
   ```

   NAPOMENA: Određivanje Fibonačijevog broja koji je veći od g bi se moglo odraditi čuvanjem samo poslednja dva Fibonačijeva broja, bez upotrebe nizova, ali ovde ga rešavamo dinamičkom realokacijom da bismo ilustrovali upotrebu funkcije realloc a i na taj način čuvamo ceo niz.

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

/* Niz ce se uvecavati u koracima od po 10 brojeva */
#define KORAK 10

int main()
{
    int *a=NULL;    /* Niz je u pocetku prazan */
    int *a_pom;
    int g;

    printf("Unesite donju granicu: \n");
    scanf("%d", &g);

    /* duzina predstavlja broj popunjenih elemenata,
       dok alocirano predstavlja broj alociranih elemenata */
    int duzina = 0, alocirano = 0;

    int i;

    do
    {
        /* Ukoliko nema vise slobodnih mesta, vrsi se prosirivanje */

        if (duzina == alocirano)
        {
            /* Niz se prosiruje za 10 elemenata vise */
            alocirano = alocirano + KORAK;

            a_pom = (float *) realloc(a, alocirano*sizeof(float));

            if (a_pom != NULL)
               a = a_pom;
            else
               {
                   free(a);
                   exit 1;
               }
        }

        /* Izracunavamo tekuci element niza (na poziciji duzina)
           po definiciji Fibonacijevog niza */
        if (duzina == 0)
            a[duzina] = 0;
        else if (duzina == 1)
```

```
            a[duzina] = 1;
        else
            a[duzina] = a[duzina-1] + a[duzina-2];

        /* Ako bismo hteli da ostampamo sve elemente dobijenog niza pisali bi
           (pre povecanja brojaca):

           printf("%d ", a[duzina]);

        */

        duzina++;
    } while (a[duzina-1] < g); /* Nakon naredbe duzina++, duzina pokazuje
                                  na element niza koji nije izracunat pa da
                                  bi pristupili poslednjem izracunatom
                                  elementu pisemo a[duzina-1] */

    /* Stampamo poslednji izracunat clan niza (koji zadovoljava uslov da
       je veci od n). On se nalazi na poziciji duzina-1 posto promenljiva
       duzina predstavlja ukupnu duzinu niza */
    printf("Rezultat je: %f\n", a[duzina-1]);

    /* Oslobadjamo niz a */
    free(a);
}
```

2. Napisati program koji za unete realne brojeve x i *epsilon* korišćenjem nizova izračunava približnu vrednost izraza $1/x$ (za $0 < x \leq 2$) sa zadatom tačnošću *epsilon* ($0 \leq epsilon \leq 0.01$) na sledeći način:

   - odrediti elemente nizova (a) i (c) koju su zadati sa:
     $a[0] = 1$
     $c[0] = 1 - x$
     $a[i] = a[i-1] \cdot (1 + c[i-1]), \quad \text{za} \quad i \geq 1$
     $c[i] = c[i-1] \cdot c[i-1], \quad \text{za} \quad i \geq 1$
   - za vrednost broja $1/x$ uzima se ona vrednost $a[n]$ za koju je zadovoljen uslov $-epsilon \leq a[n] - a[n-1] \leq epsilon$.

   Prostor za nizove (a) i (c) dinamički alocirati i povećavati im dužinu (u većim blokovima) dok ne bude zadovoljena tražena tačnost.

   Na primer:

   za $x = 0.5$ i $\epsilon = 0.0001$ program treba da vrati 2.00000

   za $x = 0.5$ i $\epsilon = 0.01$ program treba da vrati 1.999969

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

/* Niz ce se uvecavati u koracima od po 10 brojeva */
#define KORAK 10

int main()
{
    /* Nizovi su u pocetku prazni */
    float* a=NULL;
    float* c=NULL;
    float *a_pom, *c_pom;
    float x;
    float eps;

    printf("Unesite broj i zeljenu tacnost: \n");
    scanf("%f%f", &x, &eps);

    /* Duzina predstavlja broj popunjenih elemenata,
       dok alocirano predstavlja broj alociranih elemenata */
       int duzina = 0, alocirano = 0;

    float n = 1;   /* Pocetna vrednost razlike. Proizvoljan broj
                      veci od eps */
    int i;

    do
    {
        /* Ukoliko nema vise slobodnih mesta, vrsi se prosirivanje */

        if (duzina == alocirano)
        {
            /* Niz se prosiruje za 10 elemenata vise */
            alocirano = alocirano + KORAK;

            c_pom = (float *) realloc(c, alocirano*sizeof(float));
            a_pom = (float *) realloc(a, alocirano*sizeof(float));

            if (c_pom != NULL)
               c = c_pom;
            else
               {
                   free(c);
                   exit 1;
               }
```

```
            if (a_pom != NULL)
               a = a_pom;
            else
               {
                   free(a);
                   exit 1;
               }
        }

        /* Izracunavamo tekuci element niza (na poziciji duzina)
           po formulama iz postavke zadatka */
        if (duzina == 0)
        {
            a[duzina] = 1;
            c[duzina] = 1-x;
        }
        else
        {
            a[duzina] = a[duzina-1]*(1+c[duzina-1]);
            c[duzina] = c[duzina-1]*c[duzina-1];
        }

       /* Da bismo proverili da li smo postigli zeljenu tacnost
          racunamo apsolutnu vrednost razlike izmedju poslednje
          izracunatog clana niza i njegovog prethodnika (za sve
          elemente sem nultog (posto on nema svog prethodnika). */
       if (duzina>0)
       {
            n = a[duzina]-a[duzina-1];
            if (n<0)
                n = -n;
       }
       duzina++;
    } while ( n>eps);

    /* Stampamo poslednji izracunat clan niza (koji zadovoljava uslov da
       je razlika izmedju njega i njegovog prethodnika po apsolutnoj
       vrednosti manja od epsilon). On se (sada) nalazi na poziciji
       duzina-1 posto promenljiva duzina predstavlja ukupnu duzinu niza */
    printf("Rezultat je: %f\n", a[duzina-1]);

    /* Oslobadjamo nizove a i c */
    free(a);
    free(c);
}
```

3. Napisati program koji za datoteku čije se ime zadaje kao prvi argument komandne linije određuje i ispisuje reč koja se pojavljuje najviše puta u toj datoteci (pretpostavljamo da su reči dužine najviše 20 karaktera).

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

#define KORAK 10

/* Kako treba da pronadjemo rec koja se pojavljuje najvise puta u
   datoteci, osim cuvanja reci koje nalaze u datoteci moracemo da imamo
   podatak o tome koliko se puta svaka rec pojavila u toj datoteci. Iz
   tog razloga u programu cemo koristiti niz struktura reci */

typedef struct word{
    char rec[20];   /* Rec koja se pojavila u datoteci */
    int br;         /* Broj pojavljivanja date reci */
} word;

main()
{
    word *a = NULL;      /* Pokazivac na niz reci */
    word *a_pom;         /* Pomocni pokazivac */
    int duzina = 0, alocirano = 0;

    int i, max_i;
    char s[20];          /* Pomocna promenljiva u kojoj cuvamo
                            tekucu rec */

    FILE *in;

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

    if (in==NULL)
    {
       fprintf(stderr, "\nGreska pri otkrivanju datoteke: %s\n", argv[1]);
       exit(1);
    }

    /* Petlja koja nam sluzi da napravimo niz svih reci koje se pojavljuju
       u datoteci. Osim reci koja je procitana u nizu cuvamo i broj
       pojavljivanja te reci. */
    do
    {
        if (duzina == alocirano)
```

```
    {
        /* Niz se prosiruje za 10 elemenata vise */
        alocirano = alocirano + KORAK;

        a_pom = (word *) realloc(a, alocirano*sizeof(word));

        if (a_pom != NULL)
            a = a_pom;
         else
            {
                free(a);
                exit 1;
            }

    }

    /* Citamo tekucu rec */
    fscanf(in, "%s", s);

    /* Proveravamo da li je tekuca rec ranije bila procitana i ako
       jeste uvecavamo njen broj pojavljivanja za 1 */
    for(i=0; i<duzina; i++)
        if (!strcmp(a[i].rec, s))
            {
                a[i].br++;
                break;
            }

    /* Ako rec nije ranije bila procitana onda treba da je unesemo
       u niz. Kako prethodna petlja proverava da li se rec nalazi u
       nizu, u slucaju da se rec ne nalazi u nizu vazice i=duzina. */
    if (i == duzina)
    {
        strcpy(a[duzina].rec,s);
        a[duzina].br = 1;
        duzina++;
    }
} while (!feof(in));

/* Sada imamo formiran niz svih reci koje se pojavljuju u datoteci sa
   podatkom o broju pojavljivanja svake od njih. */

/* Odredjujemo maksimum niza a, po broju pojavljivanja reci, ali
   pamtimo indeks reci da bismo na kraju imali podatak koja se rec
   pojavljuje najvise puta. */
```

```
    /* Indeks maksimalnog elementa postavljamo na 0, tj. krecemo od
       pretpostavke da se prva rec pojavljuje najvise puta */
    max_i = 0;

    /* U petlji proveravamo da li je duzina neke od narednih reci
       veca od duzine trenutno najduze reci - odnosno reci na poziciji
       max_i */
    for(i=1; i < duzina; i++)
        if (a[i].br > a[max_i].br)
            max_i = i;

    /* Ispisujemo rec koja se pojavila najvise puta */
    printf("Rec koja se pojavljuje najvise puta je %s\n", a[max_i].rec);
}
```