Ponovljeni završni ispit iz predmeta Algoritmi i strukture podataka

3. srpnja 2008.

Napomena za sve zadatke:
- Nije dopušteno korištenje naredbe `goto`, te statičkih i globalnih varijabli.

Zadatak 1. *(8 bodova)*

Neka je binarno sortirano stablo zadano strukturom:

```
struct cv {
  int broj;
  struct cv *lijevo;
  struct cv *desno;
};

typedef struct cv cvor;
```

a) (4 boda) Napišite funkciju koja će izračunati sumu svih brojeva u onim čvorovima koji nisu listovi. Napišite primjer poziva napisane funkcije.
b) (4 boda) Napišite funkciju koja će u binarno sortirano stablo ubaciti novi broj. Ako navedeni broj već postoji u stablu, novi se čvor ne ubacuje. Napišite primjer poziva napisane funkcije.

Zadatak 2. *(8 bodova)*

Zadana je dvostruko povezana lista čija je glava `glava`, a rep `rep`. Napišite funkciju koja će iz zadane liste sve čvorove koji sadrže parne brojeve prebaciti *(bez stvaranja novih čvorova)* u dvostruko povezanu listu čija je glava `par_glava`, a rep `par_rep`, nakon čega početna lista, čija je glava `glava`, a rep `rep`, treba sadržavati samo neparne brojeve.

Prototip funkcije je:

```
void podijeli (atom **glava, atom **rep, atom **par_glava, atom** par_rep);
```

Zadatak 3. *(7 bodova)*

a) *(4 boda)* Napišite funkciju za dodavanje jednog elementa u gomilu realiziranu poljem tako da nakon dodavanja polje i dalje bude gomila.

   Funkcija ima prototip:

   ```
   void dodaj(int element, int gomila[], int n);
   ```

   gdje je `element` element koji se dodaje, `gomila` dosadašnje polje koje predstavlja gomilu i `n` broj elemenata u polju.

   Pretpostavite da je rezervirano dovoljno memorije za polje.

b) *(3 boda)* Zadan je niz brojeva: 5, 7, 2, 13, 6, 1, 8. Ilustrirajte (nacrtajte stablo nakon svake promjene) stvaranje gomile od zadanog polja brojeva algoritmom čija je složenost za najgori slučaj O(n).

Zadatak 4. *(7 bodova)*

Objektno orijentirana implementacija reda zadana je klasom:

```
typedef int tip;
struct at {
          tip element;
          struct at *sljed;
};
typedef struct at atom;

class Red {
     atom *ulaz, *izlaz;
     void obrisiRed();
  public:
     Red();
     ~Red();
     int Dodaj (tip element);
     int Skini (tip *element);
};
```

Objektno orijentirana implementacija stoga zadana je klasom:

```
class Stog{
     atom *vrh;
     void obrisiStog();
  public:
     Stog();
     ~Stog();
     int Dodaj (tip element);
     int Skini (tip *element);
};
```

Proširite navedene implementacije odgovarajućim konstruktorima ili funkcijama kako bi bilo moguće realizirati sljedeće podebljano označene retke:

```
Red red; int n;

…
Stog stog(&red, n); /*     Stvara stog koji sadrži prvih n elemenata iz
                           reda (ili sve elemente reda ako on sadrži manje
                           od n elemenata). Prvi element iz reda ide na
                           dno stoga. Sadržaj reda mora biti očuvan. */

Red *noviRed = stog.URed(n); /*    Vraća pokazivač na novi red koji se
                           sastoji od prvih (gornjih) n elemenata stoga
                           (ili od svih elemenata stoga ako on sadrži
                           manje od n elemenata) Prvi element u redu je
                           onaj koji je bio na vrhu stoga. Sadržaj stoga
                           mora biti očuvan. */
```

Napomena: prazni konstruktor, destruktor te funkcije `Dodaj` i `Skini` već su napisani i njih ne treba pisati. (*Naravno, smijete ih koristiti!*)

Rješenja

Zadatak 1.

a)

```
cvor* ubaci(cvor *korijen, int broj){
      if (korijen){
            if (broj < korijen->broj)
                  korijen->lijevo = ubaci(korijen->lijevo, broj);
            else if (broj > korijen->broj)
                  korijen->desno = ubaci(korijen->desno, broj);
      }
      else{
            korijen = (cvor*) malloc(sizeof(cvor));
            korijen->broj = broj;
            korijen->lijevo = NULL;
            korijen->desno = NULL;
      }
      return korijen;
}
```

b)

```
int suma(cvor *korijen){
  if (korijen){
      if (korijen->lijevo || korijen->desno){
          return korijen->broj + suma(korijen->lijevo) + suma(korijen->desno);
      }
      else
          return 0;
  }
  else
      return 0;
}
```

Zadatak 2.

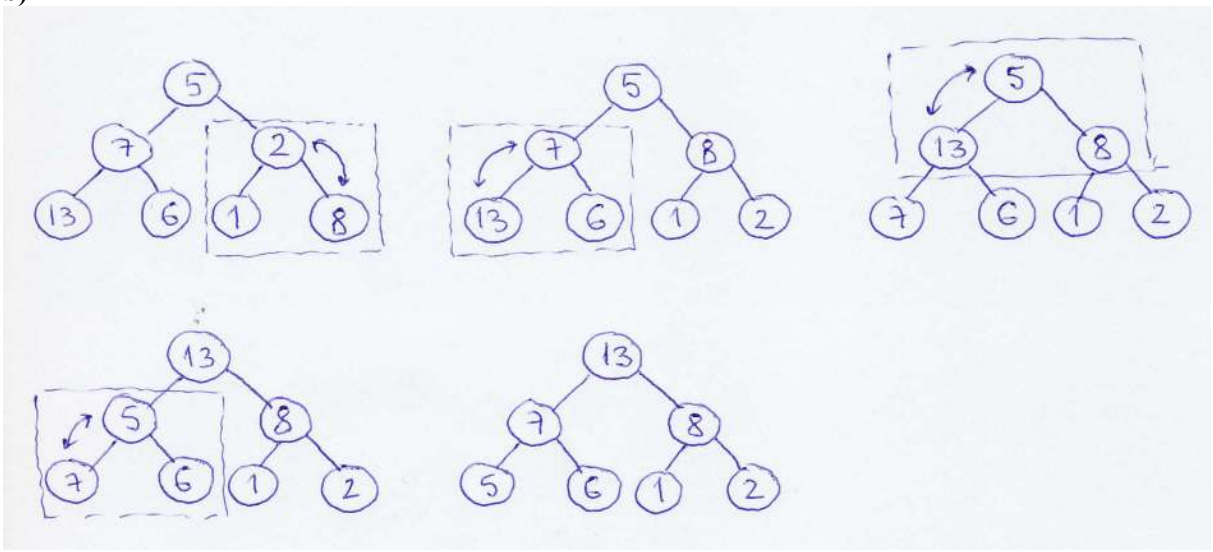
```
void podijeli (atom **glava, atom **rep, atom **par_glava, atom** par_rep) {
      atom *trenutni, *sljedeci, *prethodni;

      trenutni = *glava;

      while(trenutni){
            if (trenutni->broj % 2 == 0){
                  /* trenutnog treba "isčupati" van i
                        premjestiti u drugu listu. Stoga treba promijeniti
                        veze prethodnom i sljedećem (ako oni postoje) */
                  sljedeci = trenutni->sljed;
                  prethodni = trenutni->preth;
                  /* ako prethodnik postoji,
                        promijeni sljedbenika prethodniku */
                  if (prethodni){
                        prethodni->sljed = sljedeci;
                  }
                  else{
                        /*nema prethodnika, dakle "čupamo" van onog koji je bio
                              glava do tada, pa moramo promijeniti glavu */
                        *glava = sljedeci;
                  }

                  /* ako postoji sljedbenik, onda trenutni više nije njegov
                        prethodnik, nego je to prethodni */
                  if (sljedeci){
                        sljedeci->preth = prethodni;
                  }
                  else{
                        /* ako sljedbenik ne postoji,onda je trenutni bio rep
                        stare liste, pa to više nije, nego je to onaj ispred
                        njega */
                        *rep = prethodni;
                  }

                  /* novi prethodnik čvora kojeg "čupam" van je rep nove liste,
                        a nema sljedbenika */
                  trenutni->preth = *par_rep;
                  trenutni->sljed = NULL;

                  /* ako je postojao rep nove liste, onda je sljedbenik tog repa
                        trenutni čvor, a trenutni čvor postaje rep nove liste*/
                  if (*par_rep){
                        (*par_rep)->sljed = trenutni;
                        *par_rep = trenutni;
                  }
                  else{
                        /* ako nije bilo repa nove liste, nije bilo ni glave, pa
                              je ovaj čvor i glava i rep nove liste */
                        *par_glava = *par_rep = trenutni;
                  }

                  /* idemo na sljedećeg u listi */
                  trenutni = sljedeci;
            }
            else /* idemo na sljedećeg u listi */
                  trenutni = trenutni->sljed;
      }
}
```

Zadatak 3.

a)

```
void ubaci(int element, int gomila[], int n) {
  int i, pom, novi;
  n = n+1;
  novi = element;
  i = n/2;
  while ((i > 0) && (gomila[i] < novi)) {
    gomila[n] = gomila[i];
    n = i;
    i = i/2;
  }
  gomila[n] = novi;
}
```

b)



4. zadatak

```
Stog::Stog(Red *stari, int n){
     int i = 1;tip element;Red pom;
     this->vrh = NULL;
     while(stari->Skini(&element)){
               if (i<=n){
          pom.Dodaj(element);
          this->Dodaj(element);
          i++;
              }
              else
                pom.Dodaj(element);

     }

     while(pom.Skini(&element)){
          stari->Dodaj(element);
     }
}


Red* Stog::URed(int n){
     int i = 1;tip element;
     Red *novi = new Red();
        Stog pom;
     while(i<=n && this->Skini(&element)){
          novi->Dodaj(element);
          pom.Dodaj(element);
                 i++;
     }
     while(pom.Skini(&element)){
          this->Dodaj(element);
     }

     return novi;
}
```