Algoritmi i strukture podataka

ak. god. 2013./14.

Zadaci za 2. laboratorijsku vježbu

Tijek vježbe

Vježba počinje u terminu naznačenom u kalendaru uz poštivanje akademske četvrti. Studenti koji žele ostvariti bodove s vježbi su dužni prisutstvovati vježbi od početka svog termina do predaje vježbe.

Za svaku grupu na raspolaganju je 135 min. Studenti rješavaju zadatke zadane u nastavku ovog dokumenta i očekuje se da završe obavezni dio unutar 90 min. Posljednjih 60 min vježbe prisutan je asistent koji provjerava gotova rješenja, ispituje studente o predanom programu i o gradivu predmeta te određuje bodove. Asistent će po dolasku ispitati studente koji su gotovi s vježbom ili pričekati da prođe 90min od početka termina vježbi i tada početi ispitivati studente bez obzira na to jesu li studenti gotovi ili ne.

Studenti su nakon ispitivanja i bodovanja slobodni otići ili nastaviti rješavati dodatne zadatke.

Nadoknade

Pokušaj dolaska u drugu grupu umjesto prijavljene (radi kašnjenja, iznenadnih obaveza isl.) je za ovu vježbu **strogo zabranjen**! Razlozi su organizacijske naravi i odluka će se provoditi bez iznimaka.

Podsjećamo da dolazak na vježbe nije obavezan za prolaz na ASP-u i ne postoji prag bodova iz vježbi.

Nadoknade će biti moguće samo za studente koji dokažu objektivne okolnosti radi kojih su bili spriječeni doći na vježbe. Nadoknade će se održati tjedan nakon vježbi (9.lipnja – 13. lipnja) u formatu koji ćemo odrediti ovisno o broju zainteresiranih studenata.

Studenti koji žele nadoknaditi vježbu moraju imati liječničku ispričnicu ili drugi dokument koji jasno, nedvosmisleno i mjerodavno dokazuje objektivne razloge propuštanja termina vježbi.

Student će s dokumentom doći u sobu D365 u terminu objavljenom na stranicama predmeta. Naknadno će nakon što su evidentirani svi studenti kojima je nadoknada odobrena biti određen oblik nadoknade vježbe.

Razvojni alati

Studentima je za pisanje, prevođenje i pokretanje kreiranih programa dozvoljeno korištenje bilo kojeg razvojnog alata od onih koji su instalirani na računalima Fakulteta. Studenti su dužni sami znati koristiti alat kojeg odaberu i osoblje nije dužno pružati tehničku podršku u radu s bilo kojim od alata bez obzira je li to kombinacija uređivača teksta (Notepad, Notepad++, Pspad, vi,...) i nekog prevodioca (gcc, cl) ili neko od razvojnih okruženja (Visual Studio, CodeBlocks, CodeLite...).

Uputa za vježbu

Za vrijeme laboratorijskih vježbi dopušteno je korištenje računala isključivo u svrhu izrade labosa. Svaka zlouporaba (surfanje, chat i slično) bit će sankcionirana udaljavanjem s vježbi i adekvatnim evidentiranjem aktivnosti u nastavi.

Posebno, ne tražite od demonstratora da rješavaju zadatke kakvi se pojavljuju na blicevima: ako vam nešto nije jasno, isprobajte zadatke na računalu i sami dođite do rješenja - demonstratori će vam rado pomoći, ali neće rješavati zadatke umjesto vas!

Na laboratorijske vježbe dođite pripremljeni. Proučite dosad ispredavano gradivo i primjere sa slajdova. Programski kodovi nalaze se u repozitoriju predmeta i možete ih slobodno koristiti prilikom rješavanja zadataka za laboratorijsku vježbu.

Za ovu vježbu koristi se pripremljeni kod. Neki korisnički tipovi podataka (strukture) su već definirani i neke funkcije za rad s podacima su implementirane i nalaze se u jednoj *header*-datoteci (`datoteka.h`) i jednoj C-datoteci (`funkcije.c`). Studenti koji koriste neko od razvojnih okruženja moraju ove datoteke uvesti u projekt, studenti koji koriste prevodioc s komandne linije moraju ove datoteke navesti pri pokretanju prevođenja i povezivanja (tzv. *linkanja*).

Osim spomenute header i C datoteke koje se koriste za rješavanje zadanih zadataka, dana je i datoteka demo.c u kojoj je primjerni program koji koristi zadane funkcije. Ovaj program može se pokrenuti za provjeru funkcioniranja razvojnog *set-up*-a, a kod se može proučiti za lakše razumijevanje rada sa zadanim funkcijama. Za samu vježbu mora se kreirati nova datoteka (`main.c`) u kojoj će biti studentski kod (main funkcija i ostale tražene funkcije).

Za dohvat sadržaja pojedinih mapa u primjeru dozvoljeno je isključivo korištenje zadanih funkcija. Zadane funkcije emuliraju rad sa zamišljenim datotečnim sustavom i za to koriste sadržaj dane datoteke `podaciDatoteke.txt` koja mora biti sadržana u dirkektoriju izvršnog program. Za ispravno rješavanje zadataka nije dozvoljeno direktno ulaženje u sadržaj `podaciDatoteke.txt` iz studentskog programa.

Vježba se sastoji od tri obavezna zadatka i nekoliko dodatnih. Za bodove iz vježbe je potrebno riješiti samo obavezne zadatke, ali upućujemo studente da riješe i dodatne za bolje razumijevanje gradiva predmeta.

Opis programa

Zadana je struktura `Datoteka`, koja opisuje datoteke u širem smislu (podatkovne datoteke i mape), a definirana je sa:

```
typedef struct{
      char vrsta;
      int  velicina;
      char putanja[255+1];
} Datoteka;
```

Komponenta vrsta označava vrstu datoteke i može imati jednu od dvije vrijednosti :

`'F'` (*file*) – označava „običnu“ datoteka ili

`'D'` (*directory*) – označava mapu.

Komponenta `velicina` predstavlja veličinu datoteke u bajtovima (veličina mapa je uvijek 0, veličina datoeke može i ne mora biti 0), a komponenta `putanja`, predstavlja punu putanju (*path*) do datoteke.

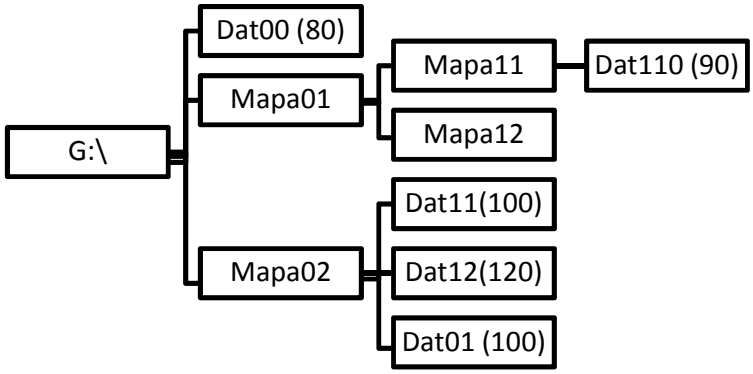
Npr. ako se datoteka `a.txt` nalazi u mapi M01 koja se nalazi na G: disku, tada `putanja` sadrži niz znakova G:\M01\a.txt. Najdulja dozvoljena putanja datoteke sadrži 255 znakova.

Za rad sa strukturom `Datoteka` dane su dvije funkcije (implementirane u funkcije.c):

- `int vratiDatoteke(char *mapa, Datoteka **datoteke);`
  Ova funkcija vraća broj mapa i datoteka koje se nalaze u mapi `mapa`, s time da se sve mape i datoteke koje se nalaze u mapi `mapa` vraćaju u polju `datoteke`. Funkcija ne vraća datoteke koje se nalaze u eventualnim podmapama mape mapa.
  Argumenti s kojima se funkcija poziva su (očito) adresa polja znakova u kojem je pohranjen niz znakova - putanja tražene mape i adresom pokazivača tipa `Datoteka *` u kojega će funkcija pohraniti adresu polja tipa `Datoteka[]` za koje je u funkciji alociran prostor na hrpi (*heap*-u). Npr. neka je zadana struktura direktorija kao na slici 1. Ako u pozivu funkcije vratiDatoteke varijabla mapa ima vrijednost `"G:"`, onda je rezultat `3`, a vraćene su datoteke (i mape) *Dat00*, *Mapa01* i *Mapa02*.
- `void prepisiDatoteku(Datoteka *odrediste,Datoteka *izvor);`
  Ova funkcija prepisuje podatke iz varijable `izvor` u varijablu `odrediste` (obje varijable su tipa `Datoteka`).

Uz strukturu `Datoteka`, implementiran je i stog pomoću polja. Stog je prilagođen upotrebi sa strukturom `Datoteka`, a implementacija sadrži funkcije

- `void init_stog(Stog *stog);` - Inicijalizacija stoga.
- `int dodaj(Datoteka stavka, Stog *stog);` - Dodavanje vrijednosti na vrh stog.
- `int skini(Datoteka *stavka, Stog *stog);` - Skidanje vrijednosti sa vrha stoga.



Slika 1. Primjer mapa i datoteka na disku. DatXX označava „obične datoteke“ (u zagradama je njihova veličina). MapaXX označava mapu.

U demo programu implementirana je rekurzivna funkcija `brojDatoteka` koja vraća koliko ima datoteka u zadanoj mapi i svim njenim podmapama. Funkcija demonstrira korištenje funkcije `vratiDatoteke`, a razumijevanje načina na koji radi može biti od koristi u rješavanju postavljenih zadataka.

Zadatak 1 *(Obavezan)*

Napravite funkciju za ispis elemenata stoga tipa `Stog`. Funkcija treba biti **neovisna o implementaciji stoga**, a nakon povratka iz funkcije stog treba biti u istom stanju u kojem je bio. Ispis na standardni izlaz treba biti formatiran, za svaki element tipa `Datoteka` treba ispisati vrstu (1 kolona), putanju (do 40 kolona, pretpostaviti da neće biti duljih putanja) i veličinu (10 kolona poravnatih desno).

Zadatak 2 *(Obavezan)*

Implementirajte rekurzivnu funkciju `najveceDatoteke`, čiji je prototip dan sa

```
void najveceDatoteke(char *mapa, Stog *s)
```

Funkcija `najveceDatoteke` treba proći kroz sve mape na disku (u početnom pozivu mapa ima vrijednost `"G:"`) i na stogu `s` za svaku viđenu mapu ostaviti podatke o datoteci (ne gledaju se mape) koja ima najveću veličinu u toj mapi (ne računajući datoteke podmapa). Pazite, u nekoj mapi ne mora nužno postojati niti jedna takva datoteka (ako je mapa prazna ili sadrži samo podmape)!

Za primjer sa slike 1, na stogu se trebaju naći datoteke *Dat00* (jedina, pa time i najveća od datoteka u mapi G:), *Dat12* (najveća u mapi *Mapa02*) i *Dat110*. U mapi *Mapa01* ne postoji najveća datoteka jer ta mapa sadrži samo podmape, dok u mapi *Mapa12* ne postoji nikakva datoteka (niti podmapa), pa samim time nema ni najveće.

Zadatak 3 *(Obavezan)*

Napišite `main` funkciju koja će korištenjem funkcija iz prvog i drugog zadatka ispisati sadržaj stoga u kojem se nalaze najveće datoteke u svakoj mapi.

Zadatak 4 *(Dodatni)*

Kreirajte funkciju `nadjiNajmanjuDatoteku` koja će na vrh stoga `s` premjestiti datoteku najmanje veličine na zadanom stogu `s`, a poredak ostalih datoteka treba ostati isti.
Ako ima više datoteka koji imaju jednaku najmanju veličinu, dovoljno je samo jednu od njih premjestiti na prvo mjesto. Prototip funkcije `nadjiNajmanjuDatoteku` je

```
void nadjiNajmanjuDatoteku(Stog *s)
```

Funkcija treba biti neovisna o implementaciji strukture podataka stog. Primjenom te funkcije nađite koja je od datoteka nađenih u drugom zadatku najmanja.

Zadatak 5 *(Dodatni)*

Kreirajte funkcije za implementaciju reda pomoću liste. Red treba biti specijaliziran za podatke tipa *`Datoteka`*. Potrebno je napraviti funkcije `init_red`, `dodaj`, `skini`. Dodatno, napravite i funkciju `ispisReda`, unutar koje će se ispisivati elementi reda. Funkcija treba biti neovisna o implementaciji strukture reda, a na izlasku iz strukture, red treba ostati nepromijenjen.

Zadatak 6 *(Dodatni)*

Pomoću funkcija za implementaciju reda, napravite funkciju `vratiSveMape`, koji će u zadanom redu `r`, vratiti sve mape koje se nalaze unutar zadane mape (uključujući i eventualne podmape). Prema primjeru sa slike 1, za zadani ulaz G: treba u redu vratiti podatke o mapama *Mapa01*, *Mapa02*, *Mapa11* i *Mapa12*.