II PARCIJALNI ISPIT IZ PREDMETA "TEHNIKE PROGRAMIRANJA" (GRUPA A)

NAPOMENA: *Sve funkcije čija je implementacija duža od dvije naredbe obavezno implementirajte izvan klase. Također, sve metode koje su inspektori obavezno deklarirajte kao takve.*

Zadatak 1 (6 poena):

Mada je već odavno u čitavom svijetu izvršena standardizacija mjernih jedinica za razne fizikalne veličine (SI sistem), Sjedinjene Američke Države se uporno opiru uvođenju ovih jedinica, nego u internoj upotrebi i dalje koriste svoje vlastite mjerne jedinice, koje su, uz neke izmjene, uglavnom naslijeđene iz Britanskog kraljevstva (interesantno je da su još jedine dvije države na svijetu pored SAD-a koje se opiru uvođenju SI sistema Liberija i Mjanmar, bivša Burma). Tako se, u SAD-u, dužine uglavnom izražavaju u *inčima* (inch), *stopama* (feet), *jardima* (yard) i *miljama* (mile). Tako, jedna stopa ima 12 inča, jedan jard ima 3 stope, dok jedna milja ima 1760 jardi (niđe veze, što bi rekli u Hercegovini). Veza sa SI sistemom lako se izvodi znajući da jedan inč iznosi 2,54 centimetra.

Vaš zadatak je da napravite klasu koja omogućava računanje sa američkim mjerama za dužinu, kao i pretvorbe između američkog i standardnog sistema mjerenja dužine. Interfejs klase treba da sadrži sljedeće elemente:

- Konstruktor sa tri cjelobrojna parametra koja omogućava zadavanje dužine u *jardima*, *stopama* i *inčima* (milje ćemo ovdje zanemariti), npr. 9 jardi, 2 stope i 7 inča. Vrijednosti ovih parametara ne moraju biti ograničene (npr. broj inča ne mora biti u opsegu od 0 do 11). Stoga je, recimo, sasvim legalno zadati 5 jardi, 7 stopa i 14 inča, ali treba imati na umu da je to isto što i 7 jardi, 2 stope i 2 inča. Kako su parametri cjelobrojni, smatraćemo da nas ne zanimaju dijelovi inča, odnosno sve dužine uvijek će imati cijeli broj inča.
- Konstruktor sa jednim realnim parametrom, koji omogućava da se dužina zada *u metrima*. Unutar ovog konstruktora potrebno je izvršiti pretvorbu metričkih u američke mjerne jedinice. Rezultat pretvorbe treba zaokružiti na cijeli broj inča.
- Metodu sa tri cjelobrojna parametra koja očitava broj jardi, stopa i inča pohranjen u objektu i smješta ih respektivno u tri navedena parametra. Pri tome, očitane vrijednosti moraju biti normalizirane u smislu da broj inča uvijek mora biti u opsegu od $0-11$ a broj stopa u opsegu $0-2$.
- Metodu koja kao rezultat daje pohranjenu dužinu *u metrima* (kao realan broj).
- Preklopljeni operator "`+`" koji daje kao rezultat novu dužinu koja je jednaka zbiru dužina koje su zadane kao operandi.
- Preklopljeni operator "`+=`" koji obezbjeđuje da izraz oblika "`X += Y`" uvijek ima isto značenje kao i izraz "`X = X + Y`".
- Preklopljeni operator "`++`" koji povećava dužinu pohranjenu u objektu za 1 inč. Potrebno je podržati i prefiksnu i postfiksnu verziju ovog operatora.
- Preklopljeni operator "`*`" koji omogućava množenje dužine sa realnim brojem odnosno množenje realnog broja sa dužinom (množenje dvije dužine ne treba podržati, jer rezultat takvog množenja nije dužina, nego površina), dajući novu dužinu kao rezultat. Rezultat se zaokružuje na cijeli broj inča.
- Preklopljeni operator "`/`" koji dijeli dvije dužine i daje realni broj kao rezultat (jednak odnosu dužina koje se dijele).
- Preklopljeni operator "`<<`" koji omogućava ispis dužina na izlazni tok u formatu "*a* `yd` *b* `ft` *z* `in`" (recimo "`9 yd 2 ft 7 in`").

VAŽNA NAPOMENA: U ovom zadatku *namjerno* nije rečeno šta su atributi klase koju treba razviti. U načelu, za atribute možete uzeti *šta god hoćete* što Vam treba da ova klasa radi ono što treba da radi (to i jeste poenta enkapsulacije i skrivanja informacija). Međutim, sama implementacija klase će *bitno ovisiti* od toga šta uzmete da su atributi. U zavisnosti od izbora, implementacija može biti izuzetno jednostavna, ali i prilično komplicirana. Zato nemojte biti brzopleti i dobro razmislite šta uzeti za atribute. Rješenje koje prvo pada na pamet nije nužno i najpametnije rješenje. Stoga, ukoliko Vam se rješavanje ovog zadatka primijetno zakomplicira, znajte da je do Vas. Isključivo do Vas.

Rješenje:

Prilikom projektiranja ove klase ključna odluka koja pojednostavljuje njeno projektiranje je da se dužina čuva *u samo jednom atributu*, izraženu u količini najmanjih mjernih jedinica (inča). Na taj način, sve se može računati isključivo u inčima (i to u cjelobrojnim vrijednostima), dok je jedino mjesto gdje treba uopće voditi računa da postoje stope i jardi metoda za čitanje broja jardi, stopa i inča, kao i operator za ispis, koji svakako može pozvati tu metodu za potrebe očitanja. Alternativno se dužina može čuvati i u *metrima*, ali se tada javlja potreba za malo više pretvorbi, uz dotatnu nepovoljnu okolnost rada sa necijelim brojevima. U svakom slučaju, realizacija se osjetno komplicira ukoliko se količina jardi, stopa i inča čuva u *posebnim atributima*, jer je tada gotovo u svakoj metodi potrebno voditi računa o toj razdvojenosti. Ovako, implementacije gotovo svih metoda su trivijalne (u jednoj naredbi), pa će biti implementirane unutar klase:

```
class ADuzine {
  int inci;
public:
  ADuzine(int jardi, int stope, int inci) {
    ADuzine::inci = 36 * jardi + 12 * stope + inci;
  }
  ADuzine(double metri) { inci = int(metri / 0.0254 + 0.5); }
  void Ocitaj(int &jardi, int &stope, int &inci);
  double DajMetre() const { return inci * 0.0254; }
  friend ADuzine operator +(ADuzine d1, ADuzine d2) {
    return ADuzine(0, 0, d1.inci + d2.inci);
  }
  ADuzine &operator +=(ADuzine d) { inci += d.inci; return *this; }
  ADuzine &operator ++() { inci++; return *this; }
  ADuzine operator ++(int) { ADuzine stara(*this); inci++; return stara; }
  friend ADuzine operator *(ADuzine d, double faktor) {
    return ADuzine(0, 0, int(d.inci * faktor + 0.5));
  }
  friend ADuzine operator *(double faktor, ADuzine d) { return d * faktor; }
  friend double operator /(ADuzine d1, ADuzine d2) {
    return double(d1.inci) / d2.inci;
  }
  friend ostream &operator <<(ostream &tok, ADuzine d);
};
```

Praktično jedini elementi klase koji zaslužuju da budu implementirani izvan klase su metoda "`Ocitaj`" i operator ispisa:

```
void ADuzine::Ocitaj(int &jardi, int &stope, int &inci) {
  jardi = ADuzine::inci / 36;
  stope = (ADuzine::inci % 36) / 12;
  inci = ADuzine::inci % 12;
}
ostream &operator <<(ostream &tok, ADuzine d) {
  int jardi, stope, inci;
  d.Ocitaj(jardi, stope, inci);
  return tok << jardi << " yd " << stope << " ft " << inci << " in";
}
```

Sada slijedi nekoliko općih komentara vezanih za implementaciju ove klase:

- Dodatni član 0.5 u konstruktoru koji pretvara metre u američke jedinice i operatoru za množenje je potreban da se dobije zaokruživanje na najbliži cijeli broj, jer obična konverzija u tip "`int`" prosto odsjeca decimale bez zaokruživanja (ovo se neće mnogo gledati pri ocjenjivanju, ali stoji da je potrebno).
- U većini funkcija objekti tipa "`ADuzine`" prenose se u funkcije *po vrijednosti*, a ne kao reference na konstantne objekte kako je uobičajeno. Naravno, upotreba referenci nije nikakva greška, ali je činjenica da su objekti tipa "`ADuzine`" toliko "lagani" sa aspekta utroška računarskih resursa (sastoje se samo od jednog cijelog broja) da njihovo prenošenje po referenci sa aspekta efikasnosti donosi više štete nego koristi.

- Zahvaljujući činjenici da parametri u konstruktoru ne moraju biti ograničeni, dužina se može zadati samo brojem inča, stavljajući broj stopi i jardi na nulu. Ovo je iskorišteno na više mjesta. U suprotnom, realizacija bi bila mnogo komplikovanija.
- Konverzija u tip "`double`" u operatoru dijeljenja je neophodna da bi se izbjeglo cjelobrojno dijeljenje.

Razumije se da se ovaj zadatak mogao riješiti na mnogo različitih načina. Međutim, ovdje prikazano rješenje je vjerovatno najkraće, najjednostavnije i najefikasnije.

Zadatak 2 (8 poena):

Razvijte klasu koja predstavlja tabelarno zadanu funkciju, tj. funkciju koja nije zadana analitički, nego kao skup parova realnih brojeva oblika (x_i, y_i) koji su dobijeni recimo mjerenjem. Ovi parovi će se čuvati u dinamički alociranom nizu objekata tipa "`Par`", pri čemu "`Par`" predstavlja običnu strukturu sa dva realna atributa "`x`" i "`y`". Interfejs ove klase treba sadržavati sljedeće elemente:

- Konstruktor sa jednim parametrom koji vrši alokaciju prostora za čuvanje parova. Parametar predstavlja inicijalno predviđeni broj parova koji se mogu pohraniti (vidjećemo da se ovaj broj može naknadno mijenjati tokom rada klase). Ovaj parametar treba imati podrazumijevanu vrijednost 30, koja se koristi ukoliko korisnik ne navede parametar. U svakom slučaju, ovaj konstruktor se ne smije koristiti za automatsku konverziju cjelobrojnih podataka u tip ove klase.
- Destruktor, koji oslobađa sve resurse koje su primjerci ove klase zauzeli tokom svog života.
- Konstruktor kopije i preklopljeni operator dodjele koji omogućavaju da se primjerci ove klase mogu bezbjedno kopirati i međusobno dodjeljivati na bazi dubokog kopiranja.
- Metodu sa dva parametra koja dodaje novi par (x, y) u postojeći skup parova. Parametri metode su upravo x i y. Ukoliko u skupu parova već postoji par čija je prva koordinata jednaka x, treba baciti izuzetak (s obzirom da nije moguće imati dvije vrijednosti funkcije y za istu vrijednost x). Ukoliko se popuni sav alocirani prostor, treba alocirati novi prostor sa 20 dodatnih mjesta, iskopirati u njega sve pohranjene parove, obrisati stari (popunjeni) prostor i nastaviti dodavanje u novi prostor.
- Metodu koja briše sve unesene parove.
- Metodu sa jednim parametrom x koja briše iz kolekcije par čija je prva koordinata x ukoliko takav postoji, a u suprotnom baca izuzetak.
- Metodu sa četiri parametra f, $xmin$, $xmax$ i dx pri čemu je f funkcija koja prima realni broj a vraća realni broj kao rezultat, dok su $xmin$, $xmax$ i dx realni brojevi. Metoda treba da u kolekciju doda sve parove oblika $(x, f(x))$ za sve vrijednosti x od $xmin$ do $xmax$ u koraku dx (tj. da tabelira funkciju f na zadanom intervalu sa zadanim korakom i da pohrani rezultate tabeliranja u kolekciju).
- Preklopljeni operator "`()`" koji vraća vrijednost funkcije u tački x (koja se zadaje kao argument) dobijenu postupkom linearne interpolacije (to je ono što se dobije kada se pretpostavi da su sve tačke (x_i, y_i) u rastućem poretku po x-ovima prosto spojene dužima). Formula za linearnu interpolaciju glasi $y = y' + (y'' - y')(x - x')/(x'' - x')$ gdje je x' najveća vrijednost među vrijednostima x_i koje su manje ili jednake od x, x'' je najmanja vrijednost među vrijednostima x_i koje su veće od x, dok su y' i y'' odgovarajuće vrijednosti y_i koje odgovaraju x' i x''. Ukoliko x' ili x'' ne postoje (npr. ako nema vrijednosti x_i koje su manje od x), interpolacija nije moguća i treba baciti izuzetak.
- Metodu koja snima čitavu kolekciju u binarnu datoteku čije se ime zadaje kao parametar, te konstruktor sa jednim parametrom koji prilikom kreiranja kolekcije obnavlja njen sadržaj iz binarne datoteke čije se ime zadaje kao parametar konstruktora.

Rješenje:

Na prvom mjestu, treba primijetiti da kolekcija sadrži niz *objekata* tipa "`Par`" a ne niz *pokazivača na objekte*. Od dodatnih atributa, potrebno je voditi evidenciju o broju pohranjenih parova (logička veličina kolekcije) i količini trenutno alociranog prostora (fizička veličina kolekcije). U prikazanoj implementaciji, struktura "`Par`" je definirana lokalno u privatnom dijelu klase, s obzirom da se koristi samo interno (naravno, definicija na globalnom nivou nije nikakva greška). Konstruktor, destruktor i metoda za brisanje čitave kolekcije su trivijalni, tako da će biti implementirani odmah unutar deklaracije klase:

```
class TabFun {
  struct Par {double x, y; };
  Par *parovi;
  int broj_parova, alocirano;
public:
  explicit TabFun(int kapacitet = 30) : broj_parova(0),
    alocirano(kapacitet), parovi(new Par[kapacitet]) {}
  ~TabFun() { delete[] parovi; }
  TabFun(const TabFun &tf);
  TabFun &operator =(const TabFun &tf);
  void DodajPar(double x, double y);
  void BrisiSve() { broj_parova = 0; }
  void Brisi(double x);
  void Tabeliraj(double f(double), double xmin, double xmax, double dx);
  double operator()(double x) const;
  void Sacuvaj(const char ime[]) const;
  TabFun(const char ime[]);
};
```

Treba primijetiti da je za brisanje čitave kolekcije dovoljno postaviti logičku veličinu kolekcije na nulu. Naime, prilikom implementacije klase, uvijek treba poći od onoga *šta korisnik vidi* i tako se ponašati. Jasno je da će nakon postavljanja logičke veličine kolekcije na nulu korisnik klase imati utisak da je ona prazna, tako da je to i najlakša implementacija. Nikakva dealokacija memorije nije potrebna. Uostalom, čitavo vrijeme dok korisnik koristi klasu, logički broj elemenata u kolekciji i fizička količina alocirane memorije gotovo nikada nisu jednake, a korisnik to uopće ne vidi. Stoga nema potrebe da nas sekira što nakon "brisanja" memorija ostaje alocirana. Naravno, nije greška ni izvršiti dealokaciju memorije, ali ako se to ispravno uradi. Na primjer, velika je greška samo obrisati memoriju, a ne predvidjeti nikakav mehanizam da se obezbijedi prostor za smještanje parova ukoliko se korisnik nakon brisanja odluči da dodaje nove elemente "na prazno".

Slijedi implementacija onih elemenata klase koji nisu implementirani unutar deklaracije klase. Na prvom mjestu, tu su konstruktor kopije i operator dodjele. Kod dubokog kopiranja, kad god imamo razdvojene informacije o logičkoj i fizičkoj veličini, javlja se dilema da li u odredišnom objektu alocirati onoliko prostora *koliko zaista treba* (tj. prema *logičkoj veličini*) izvornog objekta, ili onoliko *koliko je prostora alocirano* u izvornom objektu (tj. prema njegovoj *fizičkoj veličini*). Obje su varijante ispravne, ali svaka ima svoje "za i protiv". Naravno, u operatoru dodjele je poželjno izbjeći realokaciju kada god je to moguće. Ovdje je implementirana prva varijanta:

```
TabFun::TabFun(const TabFun &tf) : broj_parova(tf.broj_parova),
  alocirano(tf.broj_parova), parovi(new Par[tf.alocirano]) {
    copy(tf.parovi, tf.parovi + broj_parova, parovi);
}
TabFun &TabFun::operator =(const TabFun &tf) {
  if(alocirano < tf.broj_parova) {
    delete[] parovi; parovi = new Par[tf.broj_parova];
  }
  alocirano = broj_parova = tf.broj_parova;
  copy(tf.parovi, tf.parovi + broj_parova, parovi);
  return *this;
}
```

Metoda za dodavanje novog para je posebno interesantna zbog činjenice da implementira koncept "rasta na zahtjev" koji postoji recimo u tipovima podataka "`vector`" i "`string`", odnosno veličina kolekcije nije unaprijed ograničena, nego se može povećavati po potrebi, sve dok ima raspoložive memorije. Koncept je posve jednostavno implementirati, kao što se vidi iz priloženog koda. Treba primijetiti način kako je elegantno inicijaliziran objekat-struktura tipa "`Par`", bez obzira što ne posjeduje konstruktor. Naravno, ručna neovisna inicijalizacija atributa "`x`" i "`y`" nije nikakva greška, ali uvijek treba učiti kako može bolje i jednostavnije:

```
void TabFun::DodajPar(double x, double y) {
  for(int i = 0; i < broj_parova; i++)
    if(parovi[i].x == x) throw "Vec postoji par sa zadanom vrijednoscu x!";
  if(broj_parova >= alocirano) {
    Par *novi_prostor(new Par[alocirano + 20]);
    copy(parovi, parovi + broj_parova, novi_prostor);
    delete[] parovi;
    parovi = novi_prostor; alocirano += 20;
  }
  Par p = {x, y};
  parovi[broj_parova++] = p;
}
```

Slično kao i u metodi za brisanje čitave kolekcije, nema nikakve potrebe da se "izbrisani" par zaista fizički uklanja iz kolekcije. To zapravo čak nije ni lako uraditi, s obzirom da parovi nisu dinamički alocirani (trebalo bi alocirati novi niz, u njega prepisati sve elemente osim onog koji brišemo i nakon toga obrisati original). Čak i da su parovi dinamički alocirani, bila bi greška tek tako osloboditi mjesto za njega, a ne predvidjeti mogućnost da se to mjesto ponovo iskoristi kada se dodaje novi par (što nije tako jednostavno). Znatno je jednostavnije brisanje izvesti tako što ćemo prosto na mjesto para koji brišemo prebaciti posljednji element kolekcije i smanjiti logičku veličinu kolekcije za 1. Zauzeće memorije neće biti smanjeno (veoma je bitno uočiti da to uopće nije bitno), ali korisnik klase će dobiti željeni efekat. Što se tiče tabeliranja, njega je trivijalno izvesti, s obzirom da metoda za dodavanje para obavlja sve što je potrebno:

```
void TabFun::Brisi(double x) {
  for(int i = 0; i < broj_parova; i++)
    if(parovi[i].x == x) {
      parovi[i] = parovi[--broj_parova];
      return;
    }
  throw "Ne postoji zadana vrijednost x!";
}
void TabFun::Tabeliraj(double f(double), double xmin, double xmax,
  double dx) {
    for(double x = xmin; x <= xmax; x += dx) DodajPar(x, f(x));
}
```

Jedini izazov sa aspekta razmišljanja predstavlja izvedba operatora "()", s obzirom da pronalaženje vrijednosti *x*' i *x*" može biti praćeno izvjesnim poteškoćama. Naime, lako je sastaviti "rješenja" koja rade ispravno gotovo uvijek, ali "padaju" na nekim graničnim slučajevima. Problemi se uglavnom svode na to kako inicijalizirati vrijednosti od kojih počinje pretraga (koja se, naravno, izvodi u petlji) da bismo imali garanciju da će vrijednosti *x*' i *x*" biti uvijek pronađene ukoliko postoje, kao i da će njihovo eventualno nepostojanje biti ispravno detektirano. Ovdje prikazano rješenje je najvjerovatnije najjednostavnije, ali ni ono nije sasvim "bez mana" – naime, ono polazi od pretpostavke da će sve vrijednosti *x*-a ležati u nekom razumnom rasponu, za koji je ovdje pretpostavljeno da je od -10^{100} do 10^{100} (mislim da će se svi složiti u osnovanost ove pretpostavke). Moglo bi se i ovo ograničenje ukloniti, ali po cijenu uvođenja dodatnih logičkih varijabli i petlji za pronalaženje najveće i najmanje vrijednosti x_i. Napomenimo da rješenja zasnovana na sortiranju nisu neispravna, ali se ne mogu pohvaliti niti efikasnošću, niti elegancijom.

```
double TabFun::operator()(double x) const {
  const double OGROMAN(1e100);
  double x1(-OGROMAN), x2(OGROMAN), y1, y2;
  for(int i = 0; i < broj_parova; i++) {
    if(parovi[i].x <= x && parovi[i].x >= x1) {
      x1 = parovi[i].x; y1 = parovi[i].y;
    }
    if(parovi[i].x > x && parovi[i].x <= x2) {
      x2 = parovi[i].x; y2 = parovi[i].y;
    }
  }
  if(x1 == -OGROMAN || x2 == OGROMAN) throw "Interpolacija nije moguca!";
  return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
}
```

Smještanje sadržaja klase u binarnu datoteku i njeno obnavljanje je rutinski posao, pod uvjetom da tačno identificiramo šta je uopće potrebno sačuvati odnosno obnoviti. Ne smijemo zaboraviti da konstruktor ne smije baciti izuzetak a da prethodno ne oslobodi sav prostor koji je eventualno zauzeo u procesu kreiranja objekta:

```
void TabFun::Sacuvaj(const char ime[]) const {
  ofstream izlaz(ime, ios::out | ios::binary);
  if(!izlaz) throw "Problem pri kreiranju datoteke!";
  izlaz.write((char*)this, sizeof(TabFun));
  izlaz.write((char*)parovi, broj_parova * sizeof(Par));
  if(!izlaz) throw "Problem pri upisu u datoteku!";
}
TabFun::TabFun(const char ime[]) {
  ifstream ulaz(ime, ios::in | ios::binary);
  if(!ulaz) throw "Problem pri kreiranju datoteke!";
  ulaz.read((char*)this, sizeof(TabFun));
  parovi = new Par[alocirano];
  ulaz.read((char*)parovi, broj_parova * sizeof(Par));
  if(!ulaz) {
    delete[] parovi;
    throw "Problem pri citanju iz datoteke!";
  }
}
```

Zadatak 3 (6 poena):

Neka je *predmet* objekat koji je, između ostalog, okarakteriziran svojom specifičnom težinom (gustinom). *Kocka* je specijalna vrsta predmeta okarakterizirana dužinom stranice, dok je *Valjak* specijalna vrsta predmeta okarakteriziran poluprečnikom osnovice i visinom. Razvijte hijerarhiju klasa koje opisuju ove objekte. Apstraktna bazna klasa "`Predmet`" posjedovaće atribut koji predstavlja specifičnu težinu predmeta (realni broj), konstruktor koji inicijalizira ovaj atribut, čisto virtuelnu metodu "`DajZapreminu`" koja daje zapreminu predmeta, te metodu "`DajTezinu`" koja daje težinu predmeta (specifična težina pomnožena sa zapreminom). Klasa "`Kocka`" nasljeđuje se iz klase "`Predmet`", a posjeduje dodatni atribut koji predstavlja dužinu stranice kocke. Konstruktor ove klase ima dodatni parametar (dužinu stranice a) i izmijenjenu metodu "`DajZapreminu`", koja implementira računanje zapremine za kocku ($V = a^3$). Klasa "`Valjak`" se također nasljeđuje iz klase "`Predmet`", a posjeduje dodatne atribute koji predstavljaju poluprečnik osnovice (r) i visinu (h). Ova klasa također ima dodatne parametre u konstruktoru (r i h) i implementiranu metodu za računanje zapremine ($V = hr^2\pi$). Napisane klase demonstrirajte u testnom programu koji čita podatke o predmetima iz tekstualne datoteke u vektor, sortira predmete po ukupnoj težini u rastući poredak i na kraju ispisuje težine predmeta nakon sortiranja. Svaki red datoteke sadrži podatke o jednom predmetu, gdje početno slovo "`K`" označava kocku, a "`V`" valjak. Recimo, "`K0.75 3.5`" predstavlja kocku specifične težine 0.75 sa stranicom dužine 3.5, dok "`V1.3 4 2.4`" predstavlja valjak specifične težine 1.3 sa poluprečnikom osnovice 4 i visinom 2.4. Pretpostavite da datoteka sadrži isključivo ispravne podatke.

Rješenje:

Počećemo od apstraktne bazne klase. Mada u ovom primjeru nije striktno neophodno, programer koji imalo misli na budućnost uvijek će u baznu klasu staviti virtuelni destruktor koji ne radi ništa:

```
class Predmet {
  double spec_tezina;
public:
  Predmet(double spec_tezina) : spec_tezina(spec_tezina) {}
  virtual ~Predmet() {}
  virtual double DajZapreminu() const = 0;
  double DajTezinu() const { return spec_tezina * DajZapreminu(); }
};
```

Izvedenim klasama samo ostaje da konkretiziraju ono što je u baznoj klasi ostalo apstraktno:

```
class Kocka : public Predmet {
  double a;
public:
  Kocka(double spec_tezina, double a) : Predmet(spec_tezina), a(a) {}
  double DajZapreminu() const { return a * a * a; }
};
class Valjak : public Predmet {
  double r, h;
public:
  Valjak(double spec_tezina, double r, double h) : Predmet(spec_tezina),
    r(r), h(h) {}
  double DajZapreminu() const { return h * r * r * M_PI; }
};
```

Potrebna nam je i funkcija kriterija za sortiranje. Radi polimorfizma, ona mora raditi sa pokazivačima:

```
bool Kriterij(const Predmet *p1, const Predmet *p2) {
  return p1->DajTezinu() < p2->DajTezinu();
}
```

Preostaje još glavni program. Jasno je da u vektoru treba čuvati pokazivače na predmete umjesto samih predmeta, dok će se sami predmeti dinamički alocirati. Naime, jedino na taj način je moguće postići polimorfno ponašanje (ako ne računamo mogućnost postizanja polimorfizma korištenjem posebnih specijalno razvijenih klasa za upravljanje, kao u Predavanju 14). Naravno, sami objekti se tada moraju dinamički kreirati (ne zaboravimo ih obrisati na kraju rada).:

```
int main() {
  ifstream ulaz("TEST.TXT");
  vector<Predmet*> predmeti;
  for(;;) {
    char znak;
    double spec_tezina;
    ulaz >> znak >> spec_tezina;
    if(!ulaz) break;
    if(znak == 'K') {
      double a;
      ulaz >> a;
      predmeti.push_back(new Kocka(spec_tezina, a));
    }
    else if(znak == 'V') {
      double r, h;
      ulaz >> r >> h;
      predmeti.push_back(new Valjak(spec_tezina, r, h));
    }
  }
  sort(predmeti.begin(), predmeti.end(), Kriterij);
  for(int i = 0; i < predmeti.size(); i++)
    cout << predmeti[i]->DajTezinu();
  for(int i = 0; i < predmeti.size(); i++) delete predmeti[i];
}
```

II PARCIJALNI ISPIT IZ PREDMETA "TEHNIKE PROGRAMIRANJA" (GRUPA B)

NAPOMENA: *Sve funkcije čija je implementacija duža od dvije naredbe obavezno implementirajte izvan klase. Također, sve metode koje su inspektori obavezno deklarirajte kao takve.*

Zadatak 1 (6 poena):

Mada je već odavno u čitavom svijetu izvršena standardizacija mjernih jedinica za razne fizikalne veličine (SI sistem), Sjedinjene Američke Države se uporno opiru uvođenju ovih jedinica, nego u internoj upotrebi i dalje koriste svoje vlastite mjerne jedinice, koje su, uz neke izmjene, uglavnom naslijeđene iz Britanskog kraljevstva (interesantno je da su još jedine dvije države na svijetu pored SAD-a koje se opiru uvođenju SI sistema Liberija i Mjanmar, bivša Burma). Tako se, u SAD-u, zapremina tečnosti uglavnom izražava u *uncama* (ounce), *galonima* (gallon) i *barelima* (barrel). Tako, jedan galon ima 128 unca, dok jedan barel ima 42 galona (ovo se odnosi na naftu – da stvar bude još besmislenija, koliko iznosi jedan barel ovisi i od vrste tečnosti, tako da recimo barel piva ima 31 galon dok barel viskija ima 40 galona, ali to ćemo za potrebe ovog zadatka zanemariti). Veza sa SI sistemom lako se izvodi znajući da jedna unca iznosi 0,02957353 litara (litar je u suštini kubni decimetar).

Vaš zadatak je da napravite klasu koja omogućava računanje sa američkim mjerama za zapreminu tečnosti, kao i pretvorbe između američkog i standardnog sistema mjerenja zapremine tečnosti. Interfejs klase treba da sadrži sljedeće elemente:

- Konstruktor sa tri cjelobrojna parametra koja omogućava zadavanje zapremine tečnosti u *barelima*, *galonima* i *uncama*, npr. 3 barela, 12 galona i 56 unca. Vrijednosti ovih parametara ne moraju biti ograničene (npr. broj galona ne mora biti u opsegu od 0 do 41). Stoga je, recimo, sasvim legalno zadati 3 barela, 90 galona i 150 unca, ali treba imati na umu da je to isto što i 5 barela, 7 galona i 22 unce. Kako su parametri cjelobrojni, smatraćemo da nas ne zanimaju dijelovi unce, odnosno sve zapremine uvijek će imati cijeli broj unca.
- Konstruktor sa jednim realnim parametrom, koji omogućava da se zapremina tečnosti zada *u litrima*. Unutar ovog konstruktora potrebno je izvršiti pretvorbu standardnih u američke mjerne jedinice. Rezultat pretvorbe treba zaokružiti na cijeli broj unci.
- Metodu sa tri cjelobrojna parametra koja očitava broj barela, galona i unci pohranjen u objektu i smješta ih redom u tri navedena parametra. Pri tome, očitane vrijednosti moraju biti normalizirane u smislu da broj unci uvijek mora biti u opsegu od $0-127$ a broj galona u opsegu $0-41$.
- Metodu koja kao rezultat daje pohranjenu zapreminu tečnosti *u litrima* (kao realan broj).
- Preklopljeni operator "`+`" koji daje kao rezultat novu zapreminu koja je jednaka zbiru zapremina koje su zadane kao operandi.
- Preklopljeni operator "`+=`" koji obezbjeđuje da izraz oblika "`X += Y`" uvijek ima isto značenje kao i izraz "`X = X + Y`".
- Preklopljeni operator "`++`" koji povećava zapreminu pohranjenu u objektu za 1 uncu. Potrebno je podržati i prefiksnu i postfiksnu verziju ovog operatora.
- Preklopljeni operator "`*`" koji omogućava množenje zapremine sa realnim brojem odnosno množenje realnog broja sa zapreminom (množenje dvije zapremine ne treba podržati, jer rezultat takvog množenja nema fizikalni smisao), dajući novu zapreminu kao rezultat. Rezultat se zaokružuje na cijeli broj unci.
- Preklopljeni operator "`/`" koji dijeli dvije zapremine i daje realni broj kao rezultat (jednak odnosu zapremina koje se dijele).
- Preklopljeni operator "`<<`" koji omogućava ispis zapremine tečnosti na izlazni tok u formatu "*a* `bbl` *b* `gal` *z* `oz`" (recimo "`3 bbl 12 gal 56 oz`").

VAŽNA NAPOMENA: U ovom zadatku *namjerno* nije rečeno šta su atributi klase koju treba razviti. U načelu, za atribute možete uzeti *šta god hoćete* što Vam treba da ova klasa radi ono što treba da radi (to i jeste poenta enkapsulacije i skrivanja informacija). Međutim, sama implementacija klase će *bitno ovisiti* od toga šta uzmete da su atributi. U zavisnosti od izbora, implementacija može biti izuzetno jednostavna, ali i prilično komplicirana. Zato nemojte biti brzopleti i dobro razmislite šta uzeti za atribute. Rješenje koje prvo pada na pamet nije nužno i najpametnije rješenje. Stoga, ukoliko Vam se rješavanje ovog zadatka primijetno zakomplicira, znajte da je do Vas. Isključivo do Vas.

Rješenje:

Prilikom projektiranja ove klase ključna odluka koja pojednostavljuje njeno projektiranje je da se dužina čuva *u samo jednom atributu*, izraženu u količini najmanjih mjernih jedinica (unca). Na taj način, sve se može računati isključivo u uncama (i to u cjelobrojnim vrijednostima), dok je jedino mjesto gdje treba uopće voditi računa da postoje bareli i galoni metoda za čitanje broja barela, galona i unca, kao i operator za ispis, koji svakako može pozvati tu metodu za potrebe očitanja. Alternativno se zapremina može čuvati i u *litrima*, ali se tada javlja potreba za malo više pretvorbi, uz dotatnu nepovoljnu okolnost rada sa necijelim brojevima. U svakom slučaju, realizacija se osjetno komplicira ukoliko se količina barela, galona i unci čuva u *posebnim atributima*, jer je tada gotovo u svakoj metodi potrebno voditi računa o toj razdvojenosti. Ovako, implementacije gotovo svih metoda su trivijalne (u jednoj naredbi), pa će biti implementirane unutar klase:

```
class AZapTec {
  int unce;
public:
  AZapTec(int bareli, int galoni, int unce) {
    AZapTec::unce = 5376 * bareli + 128 * galoni + unce;
  }
  AZapTec(double litri) { unce = int(litri / 0.02957353 + 0.5); }
  void Ocitaj(int &bareli, int &galoni, int &unce);
  double DajLitre() const { return unce * 0.02957353; }
  friend AZapTec operator +(AZapTec z1, AZapTec z2) {
    return AZapTec(0, 0, z1.unce + z2.unce);
  }
  AZapTec &operator +=(AZapTec z) { unce += z.unce; return *this; }
  AZapTec &operator ++() { unce++; return *this; }
  AZapTec operator ++(int) { AZapTec stara(*this); unce++; return stara; }
  friend AZapTec operator *(AZapTec z, double faktor) {
    return AZapTec(0, 0, int(z.unce * faktor + 0.5));
  }
  friend AZapTec operator *(double faktor, AZapTec z) { return z * faktor; }
  friend double operator /(AZapTec z1, AZapTec z2) {
    return double(z1.unce) / z2.unce;
  }
  friend ostream &operator <<(ostream &tok, AZapTec z);
};
```

Praktično jedini elementi klase koji zaslužuju da budu implementirani izvan klase su metoda "`Ocitaj`" i operator ispisa:

```
void AZapTec::Ocitaj(int &bareli, int &galoni, int &unce) {
  bareli = AZapTec::unce / 5376;
  galoni = (AZapTec::unce % 5376) / 128;
  unce = AZapTec::unce % 128;
}
ostream &operator <<(ostream &tok, AZapTec z) {
  int bareli, galoni, unce;
  z.Ocitaj(bareli, galoni, unce);
  return tok << bareli << " bbl " << galoni << " gal " << unce << " oz";
}
```

Sada slijedi nekoliko općih komentara vezanih za implementaciju ove klase:

- Dodatni član 0.5 u konstruktoru koji pretvara metre u američke jedinice i operatoru za množenje je potreban da se dobije zaokruživanje na najbliži cijeli broj, jer obična konverzija u tip "**`int`**" prosto odsjeca decimale bez zaokruživanja (ovo se neće mnogo gledati pri ocjenjivanju, ali stoji da je potrebno).
- U većini funkcija objekti tipa "`AZapTec`" prenose se u funkcije *po vrijednosti*, a ne kao reference na konstantne objekte kako je uobičajeno. Naravno, upotreba referenci nije nikakva greška, ali je činjenica da su objekti tipa "`AZapTec`" toliko "lagani" sa aspekta utroška računarskih resursa (sastoje se samo od jednog cijelog broja) da njihovo prenošenje po referenci sa aspekta efikasnosti donosi više štete nego koristi.

- Zahvaljujući činjenici da parametri u konstruktoru ne moraju biti ograničeni, zapremina se može zadati samo brojem unci, stavljajući broj galona i barela na nulu. Ovo je iskorišteno na više mjesta. U suprotnom, realizacija bi bila mnogo komplikovanija.
- Konverzija u tip "`double`" u operatoru dijeljenja je neophodna da bi se izbjeglo cjelobrojno dijeljenje.

Razumije se da se ovaj zadatak mogao riješiti na mnogo različitih načina. Međutim, ovdje prikazano rješenje je vjerovatno najkraće, najjednostavnije i najefikasnije.

Zadatak 2 (8 poena):

Razvijte klasu koja predstavlja tabelarno zadanu funkciju, tj. funkciju koja nije zadana analitički, nego kao skup parova realnih brojeva oblika (x_i, y_i) koji su dobijeni recimo mjerenjem. Ovi parovi će se čuvati u dinamički alociranom nizu objekata tipa "`Par`", pri čemu "`Par`" predstavlja običnu strukturu sa dva realna atributa "`x`" i "`y`". Interfejs ove klase treba sadržavati sljedeće elemente:

- Konstruktor sa jednim parametrom koji vrši alokaciju prostora za čuvanje parova. Parametar predstavlja inicijalno predviđeni broj parova koji se mogu pohraniti (vidjećemo da se ovaj broj može naknadno mijenjati tokom rada klase). Ovaj parametar treba imati podrazumijevanu vrijednost 50, koja se koristi ukoliko korisnik ne navede parametar. U svakom slučaju, ovaj konstruktor se ne smije koristiti za automatsku konverziju cjelobrojnih podataka u tip ove klase.
- Destruktor, koji oslobađa sve resurse koje su primjerci ove klase zauzeli tokom svog života.
- Konstruktor kopije i preklopljeni operator dodjele koji omogućavaju da se primjerci ove klase mogu bezbjedno kopirati i međusobno dodjeljivati na bazi dubokog kopiranja.
- Metodu sa dva parametra koja dodaje novi par (x, y) u postojeći skup parova. Parametri metode su upravo x i y. Ukoliko u skupu parova već postoji par čija je prva koordinata jednaka x, treba baciti izuzetak (s obzirom da nije moguće imati dvije vrijednosti funkcije y za istu vrijednost x). Ukoliko se popuni sav alocirani prostor, treba alocirati novi prostor dvostruko većeg kapaciteta, iskopirati u njega sve pohranjene parove, obrisati stari (popunjeni) prostor i nastaviti dodavanje u novi prostor.
- Metodu koja briše sve unesene parove.
- Metodu sa jednim parametrom x koja briše iz kolekcije par čija je prva koordinata x ukoliko takav postoji, a u suprotnom baca izuzetak.
- Metodu sa četiri parametra f, $xmin$, $xmax$ i dx pri čemu je f funkcija koja prima realni broj a vraća realni broj kao rezultat, dok su $xmin$, $xmax$ i dx realni brojevi. Metoda treba da u kolekciju doda sve parove oblika $(x, f(x))$ za sve vrijednosti x od $xmin$ do $xmax$ u koraku dx (tj. da tabelira funkciju f na zadanom intervalu sa zadanim korakom i da pohrani rezultate tabeliranja u kolekciju).
- Preklopljeni operator "`()`" koji vraća vrijednost funkcije u tački x (koja se zadaje kao argument) dobijenu postupkom linearne interpolacije (to je ono što se dobije kada se pretpostavi da su sve tačke (x_i, y_i) u rastućem poretku po x-ovima prosto spojene dužima). Formula za linearnu interpolaciju glasi $y = y' + (y'' - y')(x - x')/(x'' - x')$ gdje je x' najveća vrijednost među vrijednostima x_i koje su manje ili jednake od x, x'' je najmanja vrijednost među vrijednostima x_i koje su veće od x, dok su y' i y'' odgovarajuće vrijednosti y_i koje odgovaraju x' i x''. Ukoliko x' ili x'' ne postoje (npr. ako nema vrijednosti x_i koje su manje od x), interpolacija nije moguća i treba baciti izuzetak.
- Metodu koja snima čitavu kolekciju u binarnu datoteku čije se ime zadaje kao parametar, te konstruktor sa jednim parametrom koji prilikom kreiranja kolekcije obnavlja njen sadržaj iz binarne datoteke čije se ime zadaje kao parametar konstruktora.

Rješenje:

Na prvom mjestu, treba primijetiti da kolekcija sadrži niz *objekata* tipa "`Par`" a ne niz *pokazivača na objekte*. Od dodatnih atributa, potrebno je voditi evidenciju o broju pohranjenih parova (logička veličina kolekcije) i količini trenutno alociranog prostora (fizička veličina kolekcije). U prikazanoj implementaciji, struktura "`Par`" je definirana lokalno u privatnom dijelu klase, s obzirom da se koristi samo interno (naravno, definicija na globalnom nivou nije nikakva greška). Konstruktor, destruktor i metoda za brisanje čitave kolekcije su trivijalni, tako da će biti implementirani odmah unutar deklaracije klase:

```
class TabFun {
  struct Par {double x, y; };
  Par *parovi;
  int broj_parova, alocirano;
public:
  explicit TabFun(int kapacitet = 50) : broj_parova(0),
    alocirano(kapacitet), parovi(new Par[kapacitet]) {}
  ~TabFun() { delete[] parovi; }
  TabFun(const TabFun &tf);
  TabFun &operator =(const TabFun &tf);
  void DodajPar(double x, double y);
  void BrisiSve() { broj_parova = 0; }
  void Brisi(double x);
  void Tabeliraj(double f(double), double xmin, double xmax, double dx);
  double operator()(double x) const;
  void Sacuvaj(const char ime[]) const;
  TabFun(const char ime[]);
};
```

Treba primijetiti da je za brisanje čitave kolekcije dovoljno postaviti logičku veličinu kolekcije na nulu. Naime, prilikom implementacije klase, uvijek treba poći od onoga *šta korisnik vidi* i tako se ponašati. Jasno je da će nakon postavljanja logičke veličine kolekcije na nulu korisnik klase imati utisak da je ona prazna, tako da je to i najlakša implementacija. Nikakva dealokacija memorije nije potrebna. Uostalom, čitavo vrijeme dok korisnik koristi klasu, logički broj elemenata u kolekciji i fizička količina alocirane memorije gotovo nikada nisu jednake, a korisnik to uopće ne vidi. Stoga nema potrebe da nas sekira što nakon "brisanja" memorija ostaje alocirana. Naravno, nije greška ni izvršiti dealokaciju memorije, ali ako se to ispravno uradi. Na primjer, velika je greška samo obrisati memoriju, a ne predvidjeti nikakav mehanizam da se obezbijedi prostor za smještanje parova ukoliko se korisnik nakon brisanja odluči da dodaje nove elemente "na prazno".

Slijedi implementacija onih elemenata klase koji nisu implementirani unutar deklaracije klase. Na prvom mjestu, tu su konstruktor kopije i operator dodjele. Kod dubokog kopiranja, kad god imamo razdvojene informacije o logičkoj i fizičkoj veličini, javlja se dilema da li u odredišnom objektu alocirati onoliko prostora *koliko zaista treba* (tj. prema *logičkoj veličini*) izvornog objekta, ili onoliko *koliko je prostora alocirano* u izvornom objektu (tj. prema njegovoj *fizičkoj veličini*). Obje su varijante ispravne, ali svaka ima svoje "za i protiv". Naravno, u operatoru dodjele je poželjno izbjeći realokaciju kada god je to moguće. Ovdje je implementirana prva varijanta:

```
TabFun::TabFun(const TabFun &tf) : broj_parova(tf.broj_parova),
  alocirano(tf.broj_parova), parovi(new Par[tf.alocirano]) {
    copy(tf.parovi, tf.parovi + broj_parova, parovi);
}
TabFun &TabFun::operator =(const TabFun &tf) {
  if(alocirano < tf.broj_parova) {
    delete[] parovi; parovi = new Par[tf.broj_parova];
  }
  alocirano = broj_parova = tf.broj_parova;
  copy(tf.parovi, tf.parovi + broj_parova, parovi);
  return *this;
}
```

Metoda za dodavanje novog para je posebno interesantna zbog činjenice da implementira koncept "rasta na zahtjev" koji postoji recimo u tipovima podataka "`vector`" i "`string`", odnosno veličina kolekcije nije unaprijed ograničena, nego se može povećavati po potrebi, sve dok ima raspoložive memorije. Koncept je posve jednostavno implementirati, kao što se vidi iz priloženog koda. Treba primijetiti način kako je elegantno inicijaliziran objekat-struktura tipa "`Par`", bez obzira što ne posjeduje konstruktor. Naravno, ručna neovisna inicijalizacija atributa "`x`" i "`y`" nije nikakva greška, ali uvijek treba učiti kako može bolje i jednostavnije:

```
void TabFun::DodajPar(double x, double y) {
  for(int i = 0; i < broj_parova; i++)
    if(parovi[i].x == x) throw "Vec postoji par sa zadanom vrijednoscu x!";
  if(broj_parova >= alocirano) {
    Par *novi_prostor(new Par[alocirano * 2]);
    copy(parovi, parovi + broj_parova, novi_prostor);
    delete[] parovi;
    parovi = novi_prostor; alocirano *= 2;
  }
  Par p = {x, y};
  parovi[broj_parova++] = p;
}
```

Slično kao i u metodi za brisanje čitave kolekcije, nema nikakve potrebe da se "izbrisani" par zaista fizički uklanja iz kolekcije. To zapravo čak nije ni lako uraditi, s obzirom da parovi nisu dinamički alocirani (trebalo bi alocirati novi niz, u njega prepisati sve elemente osim onog koji brišemo i nakon toga obrisati original). Čak i da su parovi dinamički alocirani, bila bi greška tek tako osloboditi mjesto za njega, a ne predvidjeti mogućnost da se to mjesto ponovo iskoristi kada se dodaje novi par (što nije tako jednostavno). Znatno je jednostavnije brisanje izvesti tako što ćemo prosto na mjesto para koji brišemo prebaciti posljednji element kolekcije i smanjiti logičku veličinu kolekcije za 1. Zauzeće memorije neće biti smanjeno (veoma je bitno uočiti da to uopće nije bitno), ali korisnik klase će dobiti željeni efekat. Što se tiče tabeliranja, njega je trivijalno izvesti, s obzirom da metoda za dodavanje para obavlja sve što je potrebno:

```
void TabFun::Brisi(double x) {
  for(int i = 0; i < broj_parova; i++)
    if(parovi[i].x == x) {
      parovi[i] = parovi[--broj_parova];
      return;
    }
  throw "Ne postoji zadana vrijednost x!";
}
void TabFun::Tabeliraj(double f(double), double xmin, double xmax,
  double dx) {
    for(double x = xmin; x <= xmax; x += dx) DodajPar(x, f(x));
}
```

Jedini izazov sa aspekta razmišljanja predstavlja izvedba operatora "()", s obzirom da pronalaženje vrijednosti *x*' i *x*" može biti praćeno izvjesnim poteškoćama. Naime, lako je sastaviti "rješenja" koja rade ispravno gotovo uvijek, ali "padaju" na nekim graničnim slučajevima. Problemi se uglavnom svode na to kako inicijalizirati vrijednosti od kojih počinje pretraga (koja se, naravno, izvodi u petlji) da bismo imali garanciju da će vrijednosti *x*' i *x*" biti uvijek pronađene ukoliko postoje, kao i da će njihovo eventualno nepostojanje biti ispravno detektirano. Ovdje prikazano rješenje je najvjerovatnije najjednostavnije, ali ni ono nije sasvim "bez mana" – naime, ono polazi od pretpostavke da će sve vrijednosti *x*-a ležati u nekom razumnom rasponu, za koji je ovdje pretpostavljeno da je od -10^{100} do 10^{100} (mislim da će se svi složiti u osnovanost ove pretpostavke). Moglo bi se i ovo ograničenje ukloniti, ali po cijenu uvođenja dodatnih logičkih varijabli i petlji za pronalaženje najveće i najmanje vrijednosti x_i. Napomenimo da rješenja zasnovana na sortiranju nisu neispravna, ali se ne mogu pohvaliti niti efikasnošću, niti elegancijom.

```
double TabFun::operator()(double x) const {
  const double OGROMAN(1e100);
  double x1(-OGROMAN), x2(OGROMAN), y1, y2;
  for(int i = 0; i < broj_parova; i++) {
    if(parovi[i].x <= x && parovi[i].x >= x1) {
      x1 = parovi[i].x; y1 = parovi[i].y;
    }
    if(parovi[i].x > x && parovi[i].x <= x2) {
      x2 = parovi[i].x; y2 = parovi[i].y;
    }
  }
  if(x1 == -OGROMAN || x2 == OGROMAN) throw "Interpolacija nije moguca!";
  return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
}
```

Smještanje sadržaja klase u binarnu datoteku i njeno obnavljanje je rutinski posao, pod uvjetom da tačno identificiramo šta je uopće potrebno sačuvati odnosno obnoviti. Ne smijemo zaboraviti da konstruktor ne smije baciti izuzetak a da prethodno ne oslobodi sav prostor koji je eventualno zauzeo u procesu kreiranja objekta:

```
void TabFun::Sacuvaj(const char ime[]) const {
  ofstream izlaz(ime, ios::out | ios::binary);
  if(!izlaz) throw "Problem pri kreiranju datoteke!";
  izlaz.write((char*)this, sizeof(TabFun));
  izlaz.write((char*)parovi, broj_parova * sizeof(Par));
  if(!izlaz) throw "Problem pri upisu u datoteku!";
}
TabFun::TabFun(const char ime[]) {
  ifstream ulaz(ime, ios::in | ios::binary);
  if(!ulaz) throw "Problem pri kreiranju datoteke!";
  ulaz.read((char*)this, sizeof(TabFun));
  parovi = new Par[alocirano];
  ulaz.read((char*)parovi, broj_parova * sizeof(Par));
  if(!ulaz) {
    delete[] parovi;
    throw "Problem pri citanju iz datoteke!";
  }
}
```

Zadatak 3 (6 poena):

Neka je *predmet* objekat koji je, između ostalog, okarakteriziran svojom specifičnom težinom (gustinom). *Kugla* je specijalna vrsta predmeta okarakterizirana svojim poluprečnikom, dok je *Kvadar* specijalna vrsta predmeta okarakterizirana svojim dužinama stranica. Razvijte hijerarhiju klasa koje opisuju ove objekte. Apstraktna azna klasa "`Predmet`" posjedovaće atribut koji predstavlja specifičnu težinu predmeta (realni broj), konstruktor koji inicijalizira ovaj atribut, čisto virtuelnu metodu "`DajZapreminu`" koja daje zapreminu predmeta, te metodu "`DajTezinu`" koja daje težinu predmeta (specifična težina pomnožena sa zapreminom). Klasa "`Kugla`" nasljeđuje se iz klase "`Predmet`", a posjeduje dodatni atribut koji predstavlja poluprečnik kugle. Konstruktor ove klase ima dodatni parametar (poluprečnik r) i izmijenjenu metodu "`DajZapreminu`", koja implementira računanje zapremine za kuglu ($V = 4\pi r^3/3$). Klasa "`Kvadar`" se također nasljeđuje iz klase "`Predmet`", a posjeduje dodatne atribute koji predstavljaju dužine stranica kvadra. Ova klasa također ima dodatne parametre u konstruktoru (stranice a, b i c) i implementiranu metodu za računanje zapremine ($V = abc$). Napisane klase demonstrirajte u testnom programu koji čita podatke o predmetima iz tekstualne datoteke u vektor, sortira predmete po ukupnoj težini u rastući poredak i na kraju ispisuje težine predmeta nakon sortiranja. Svaki red datoteke sadrži podatke o jednom predmetu, gdje početno slovo "`K`" označava kuglu, a "`Q`" kvadar. Recimo, "`K0.75 3.5`" predstavlja kuglu specifične težine 0.75 i poluprečnika 3.5, dok "`Q1.3 3.4 7 2.15`" predstavlja kvadar specifične težine 1.3 sa dužinom stranica 3.4, 7 i 2.15 respektivno. Pretpostavite da datoteka sadrži isključivo ispravne podatke.

Rješenje:

Počećemo od apstraktne bazne klase. Mada u ovom primjeru nije striktno neophodno, programer koji imalo misli na budućnost uvijek će u baznu klasu staviti virtuelni destruktor koji ne radi ništa:

```
class Predmet {
  double spec_tezina;
public:
  Predmet(double spec_tezina) : spec_tezina(spec_tezina) {}
  virtual ~Predmet() {}
  virtual double DajZapreminu() const = 0;
  double DajTezinu() const { return spec_tezina * DajZapreminu(); }
};
```

Izvedenim klasama samo ostaje da konkretiziraju ono što je u baznoj klasi ostalo apstraktno:

```
class Kugla : public Predmet {
  double r;
public:
  Kugla(double spec_tezina, double r) : Predmet(spec_tezina), r(r) {}
  double DajZapreminu() const { return 4 * M_PI * r * r * r / 3; }
};
class Kvadar : public Predmet {
  double a, b, c;
public:
  Kvadar(double spec_tezina, double a, double b, double c) :
    Predmet(spec_tezina), a(a), b(b), c(c) {}
  double DajZapreminu() const { return a * b * c; }
};
```

Potrebna nam je i funkcija kriterija za sortiranje. Radi polimorfizma, ona mora raditi sa pokazivačima:

```
bool Kriterij(const Predmet *p1, const Predmet *p2) {
  return p1->DajTezinu() < p2->DajTezinu();
}
```

Preostaje još glavni program. Jasno je da u vektoru treba čuvati pokazivače na predmete umjesto samih predmeta, dok će se sami predmeti dinamički alocirati. Naime, jedino na taj način je moguće postići polimorfno ponašanje (ako ne računamo mogućnost postizanja polimorfizma korištenjem posebnih specijalno razvijenih klasa za upravljanje, kao u Predavanju 14). Naravno, sami objekti se tada moraju dinamički kreirati (ne zaboravimo ih obrisati na kraju rada).:

```
int main() {
  ifstream ulaz("TEST.TXT");
  vector<Predmet*> predmeti;
  for(;;) {
    char znak;
    double spec_tezina;
    ulaz >> znak >> spec_tezina;
    if(!ulaz) break;
    if(znak == 'K') {
      double r;
      ulaz >> r;
      predmeti.push_back(new Kugla(spec_tezina, r));
    }
    else if(znak == 'Q') {
      double a, b, c;
      ulaz >> a >> b >> c;
      predmeti.push_back(new Kvadar(spec_tezina, a, b, c));
    }
  }
  sort(predmeti.begin(), predmeti.end(), Kriterij);
  for(int i = 0; i < predmeti.size(); i++)
    cout << predmeti[i]->DajTezinu();
  for(int i = 0; i < predmeti.size(); i++) delete predmeti[i];
}
```