Zadaća 5.

Ova zadaća nosi ukupno 6 poena. Prvi zadatak nosi 1,2 poena, drugi zadatak nosi 1 poen, treći zadatak nosi 1,1 poen, četvrti zadatak nosi 0,7 poena, dok peti i šesti zadatak nose po 1 poen. Svi zadaci traže poznavanje prvih 13 predavanja i pretpostavljeno predznanje iz predmeta "Osnove računarstva", dok peti i šesti zadatak, kao i dio trećeg zadatka traže poznavanje i Predavanja 14. Rok za predaju ove zadaće je subota, 22. VI 2024. do kraja dana.

1. Jedan od nedostataka jezika opće namjene kao sto je C++ u odnosu na specijalističke matematski orijentirane jezike je nepostojanje podrške za egzaktni rad sa racionalnim brojevima (razlomcima) kakav postoji npr. u programskim jezicima Wolfram (na kojem je zasnovan programski paket Wolfram Mathematica) ili Maple. Srećom, koncept klasa u jeziku C++ omogućava korisniku da sam definira nove tipove podataka koji zadovoljavaju njegove potrebe. Tako je moguće definirati novi tip podataka (klasu) koji omogućava tačan rad sa razlomcima (umjesto njihovog približnog predstavljanja preko decimalnih brojeva). Nažalost, takav tip podataka nije lako napraviti da radi posve dobro. Najveći problem je što tačan rezultat čak i vrlo prostih aritmetičkih operacija sa razlomcima čiji brojnici i nazivnici uopće nisu osobito veliki može zahtijevati izuzetno velike brojeve za reprezentaciju brojnika i nazivnika. Na primjer, već u računu

$$\frac{3728}{14611}+\frac{9823}{23717}+\frac{7465}{19434}=\frac{7094377705241}{6734446276758}$$

ni brojnik ni nazivnik rezultata na većini računara ne mogu da stane u tip "`int`" iako je sam rezultat sasvim "normalne" veličine (približno oko 1.05345). Istina je da se ovaj rezultat može aproksimirati razlomkom 133380/126613 sa greškom manjom od 10^{-12}, ili recimo razlomkom 397630891/377457190 sa greškom manjom od 10^{-18} (što je izvanredna tačnost), činjenica je da ovo ipak *nisu tačni rezultati*, pa se gubi cijela poenta. Naravno, situacija se popravlja ukoliko se odlučimo da brojnik i nazivnik čuvamo u nekom širem cjelobrojnom tipu, poput "`long long int`", ali i to ima svoje granice: brojnik i nazivnik rezultata

$$\frac{3728}{14611}+\frac{9823}{23717}+\frac{7465}{19434}+\frac{8606}{14243}+\frac{4931}{22067}=\frac{3981669383975848685651}{2116638357164443168998}$$

na većini računara ne mogu stati ni u tip "`long long int`" (ovaj rezultat je približno 1.88113). I dalje je činjenica da se ovaj rezultat može veoma dobro aproksimirati jednostavnijim razlomcima w(recimo razlomkom 552123131/293506287 sa greškom manjom od 10^{-17}), ali ponovo to je bijeg od osnovnog cilja koji želimo postići — da imamo tačne rezultate a ne aproksimacije. Problem nastaje čak i pri dužoj aritmetici s razlomcima čiji su brojnici i nazivnici sasvim mali. Recimo, suma razlomaka oblika i/i gdje i redom uzima sve prirodne brojeve od 1 do 50 izražava se razlomkom čiji brojnik ima 23 cifre.

Opisani problem rješava se na razne načine. Jedino pravo rješenje je da se brojnik i nazivnik uopće ne čuvaju u cjelobrojnim varijablama, nego da se individualne cifre brojnika i nazivnika čuvaju recimo u stringu ili vektoru. Mada se time rješava opisani problem, takva izvedba je vrlo komplicirana, jer se sve računske operacije moraju izvoditi "pješke" (kao pri ručnom izvođenju aritmetičkih operacija), cifru po cifru. Nažalost, internet je prepun loših realizacija ovakve klase, koje koriste algoritam noja — gurni glavu u pijesak (tj. ignoriraj problem, u nadi da će i problem ignorirati tebe). Kod takvih realizacija, ukoliko brojnik i nazivnik rezultata ne mogu da stanu u predviđeni tip za njihovo smještanje, rezultat je jednostavno *pogrešan*, pri čemu korisnik nema nikakvu indikaciju da je rezultat pogrešan. Recimo, kod velikog broja loših interpretacija koje koriste tip "`int`", račun

$$\frac{396679}{437786}+\frac{147419}{131439}=\frac{116677065415}{57542154054}$$

kod kojeg brojnik i nazivnik očito nisu reprezentabilni u tipu "`int`", daje na većini računarskih arhitektura pogrešan rezultat 712948423/1707579206 (ovaj rezultat je posljedica toga kako se na većini računarskih arhitektura tretira prekoračenje). Ovaj rezultat je zaista *vrlo pogrešan*, jer mu je vrijednost približno 0.41752, dok tačan rezultat približno iznosi 2.02768. Nažalost, korisnik takvih klasa (kojih je, da ponovimo, internet prepun), *nema nikakvu indikaciju da je rezultat posve neupotrebljiv*.

U ovom zadatku Vi ćete uraditi *nešto bolje*. Razvićete klasu "`Razlomak`" kod koje će se brojnik i nazivnik čuvati u atributima tipa "`long long int`" (dakle, i dalje će biti mogući problemi sa reprezentacijom rezultata), ali će sve aritmetičke operacije *bacati izuzetak ukoliko rezultat nije egzaktno reprezentabilan*, pa će, ako ništa drugo, korisnik klase *barem znati da za taj račun sa tom klasom ne može dobiti tačan rezultat* (kasnije će biti opisano kako ovo tačno postići). Mogu se naći i bolje implementacije, koje u slučaju da rezultat nije egzaktno reprezentabilan vraćaju najpribližniji reprezentabilni razlomak (tj. takav da je greška što je god moguće manja), uz mogućnost da korisnik sazna da li je dobio egzaktan rezultat ili ne. Mada bi bilo veoma lijepo napraviti i takvu klasu, za njenu implementaciju potrebno je poznavati neke tehnike iz teorije brojeva (verižne razlomke, itd.) koje ne spadaju u predznanja studenata koji će raditi ovaj zadatak.

Klasa "`Razlomak`" treba da sadrži dva privatna atributa koji čuvaju vrijednost brojnika i nazivnika razlomka. Konstruktor klase ima dva parametra, koji respektivno predstavljaju vrijednost brojnika i nazivnika razlomka. Ovaj konstruktor treba automatski da izvrsi skraćivanje eventualnih zajedničkih faktora u brojniku i nazivniku, tako da ce deklaracija poput

```
Razlomak r(10, 15);
```

dovesti do toga da ce brojnik razlomka "`r`" biti 2, a nazivnik 3. Također, konstruktor treba da obezbijedi da u slučaju da je razlomak negativan, negativni znak bude zapamćen u brojniku, a ne u nazivniku. Drugim riječima, nakon deklaracije

```
Razlomak r(2, -3);
```

vrijednost brojnika treba da bude –2, a nazivnika 3. Naravno, nakon deklaracije poput

```
Razlomak r(-2, -3);
```

vrijednosti brojnika i nazivnika treba da budu 2 i 3, jer se negativni znak krati. U slučaju da se za vrijednost nazivnika zada nula, konstruktor treba da baci izuzetak tipa "`logic_error`" uz prateći tekst "Nekorektan razlomak". Kraćenje izvršite dijeljenjem brojnika i nazivnika sa njihovim najvećim zajedničkim djeliocem (NZD). Za računanje NZD postoji mnogo loših i dobrih metoda, a jedan od najboljih (jednostavan i ujedno vrlo efikasan) je Euklidov algoritam, koji se zasniva na rekurzivnoj formuli

$$\text{NZD}(p,q) = \begin{cases} p, \text{za } q = 0 \\ \text{NZD}(q, \text{mod}(p,q)), \text{za } q \neq 0 \end{cases}$$

Ovdje "mod" označava ostatak pri dijeljenju. Ova formula se trivijalno može prevesti u rekurzivnu funkciju za računanje NZD, a podjednako je jednostavno izvesti stvari i bez rekurzije (u petlji se par (p, q) zamjenjuje sa parom $(q, \text{mod}(p,q))$ dok q ne postane 0, p je tada traženi NZD). U svakom slučaju, pošto će računanje NZD trebati u ovom zadatku na više mjesta, njegovo računanje povjerite privatnoj statičkoj funkciji.

Konstruktor treba također da ima podrazumijevane parametre koji omogućavaju da se razlomak zada samo jednim parametrom ili sasvim bez parametara. Ukoliko se zada sa samo jednim parametrom, taj parametar predstavlja brojnik, a nazivnik se tada postavlja na jedinicu. Treba omogućiti da se putem ovog konstruktora vrši i automatska pretvorba cijelih brojeva u objekte tipa "`Razlomak`". Konačno, ukoliko se parametri skroz izostave, kreira se nul-razlomak (brojnik 0, nazivnik 1). U svim slučajevima, mora biti podržano da se umjesto okruglih mogu koristiti i vitičaste zagrade, tj. da se može pisati "`Razlomak r{2, 3}`" umjesto "`Razlomak r(2, 3)`" odnosno "`Razlomak r{2}`" umjesto "`Razlomak r(2)`". Predvidite i trivijalne pristupne metode "`DajBrojnik`" i "`DajNazivnik`" koje vraćaju vrijednost brojnika odnosno nazivnika respektivno.

Najvažniji dio klase su operatorske funkcije za podršku aritmetičkih operacija. Na prvom mjestu, treba podržati binarne operatore "`+`", "`-`", "`*`" i "`/`" za četiri osnovne aritmetičke operacije. Pri tome, nemojte za tu svrhu koristiti dobro poznate formule

$$\frac{p_1}{q_1} \pm \frac{p_2}{q_2} = \frac{p_1 \cdot q_2 \pm p_2 \cdot q_1}{q_1 \cdot q_2}, \qquad \frac{p_1}{q_1} \cdot \frac{p_2}{q_2} = \frac{p_1 \cdot p_2}{q_1 \cdot q_2}, \qquad \frac{p_1}{q_1} / \frac{p_2}{q_2} = \frac{p_1 \cdot q_2}{q_1 \cdot p_1}$$

Naime, mada u ove formule neosporno tačne, postoji velika mogućnost da će ove formule dovesti do prekoračenja, čak i u slučaju kada je rezultat savršeno reprezentabilan nakon što se brojnik i nazivnik skrate. Mnogo manja vjerovatnoća prekoračenja je ako se koriste formule

$$\frac{p_1}{q_1} \pm \frac{p_2}{q_2} = \frac{p_1 \cdot (q_2/r) \pm p_2 \cdot (q_1/r)}{q_1 \cdot (q_2/r)}, \qquad \frac{p_1}{q_1} \cdot \frac{p_2}{q_2} = \frac{(p_1/s) \cdot (p_2/t)}{(q_1/t) \cdot (q_2/s)}, \qquad \frac{p_1}{q_1} / \frac{p_2}{q_2} = \frac{(p_1/u) \cdot (q_2/r)}{(q_1/r) \cdot (p_2/u)}$$

gdje je $r = \text{NZD}(q_1, q_2)$, $s = \text{NZD}(p_1, q_2)$, $t = \text{NZD}(p_2, q_1)$ i $u = \text{NZD}(p_1, p_2)$. Pri tome, svi podizrazi poput q_2/r itd. su *cijeli brojevi* (jer je q_2 djeljiv sa r). Na primjer, po klasičnoj formuli imamo

$$\frac{2337}{3740} + \frac{4014}{5225} = \frac{2337 \cdot 5225 + 4014 \cdot 3740}{3740 \cdot 5225} = \frac{27223185}{19541500} = \frac{44997}{32300} \quad \text{(nakon kraćenja)}$$

dok po modificiranoj formuli imamo:

$$\frac{2337}{3740} + \frac{4014}{5225} = \frac{2337 \cdot (5225/55) + 4014 \cdot (3740/55)}{3740 \cdot (5225/55)} = \frac{494967}{355300} = \frac{44997}{32300} \quad \text{(nakon kraćenja)}$$

jer je $\text{NZD}(3740, 5225) = 55$. Vidimo da se modificirane formule daju manje međurezultate, što smanjuje opasnost od prekoračenja. Ipak, ni ove modificirane formule nemaju garanciju da neće doći do prekoračenja čak i u slučajevima kada bi rezultat bio savršeno reprezentabilan (nakon kraćenja), ali ovo je maksimum koji se može postići bez korištenja komplikovanije matematike. Dakle, ovo su formule koje ćete koristiti za realizaciju aritmetičkih operacija. U svakom slučaju, rezultati svake od operacija uvijek treba da budu do kraja skraćeni (što ćete najlakše postići tako što ćete brojnik i nazivnik željenog rezultata prosto proslijediti konstruktoru, koji će obaviti neophodno skraćivanje). Ukoliko bilo koje od računanja dovede do prekoračenja u bilo kojem međurezultatu, treba baciti izuzetak tipa "`overflow_error`" uz prateći tekst "Nemoguce dobiti tacan rezultat". Ostaje pitanje kako detektirati prekoračenje. Nažalost, prekoračenje je nemoguće detektirati nakon što se već desi, moguće je jedino unaprijed utvrditi da će doći do prekoračenja. Detaljan opis kako detektirati prekoračenje u računanju kod klasične 4 aritmetičke operacije dat je u ranije objavljenom dodatku uz predavanja posvećenom prekoračenjima i neobičnostima aritmetike s nepredznačnim vrijednostima.

Pored ova četiri binarna operatora, treba podržati i unarne operatore "`+`" i "`-`". Unarni operator "`-`" vraća razlomak na koji je primijenjen sa izvrnutim znakom (uz bacanje analognog izuzetka kao u slučaju binarnih operatora ukoliko rezultat negacije nije reprezentabilan), dok unarni operator "`+`" vraća svoj operand neizmijenjen. U skladu sa programerskim bontonom, potrebno je podržati i prekopljene operatore "`+=`", "`-=`", "`*=`" i "`/=`", pri čemu izrazi "`r1 += r2`", "`r1 -= r2`", "`r1 *= r2`" i "`r1 /= r2`" treba da budu semantički ekvivalentni izrazima poput "`r1 = r1 + r2`", "`r1 = r1 + r2`", "`r1 = r1 * r2`" i "`r1 = r1 / r2`". Također, treba podržati i unarne operatore "`++`" i "`--`", pri čemu izrazi "`r++`" i "`r--`" efektivno djeluju kao izrazi "`r += 1`" i "`r -= 1`". Pri tome treba podržati kako prefiksnu, tako i postfiksnu verziju ovih operatora (postfiksne verzije vraćaju kao rezultat vrijednost razlomka prije obavljene izmjene).

Za poređenje objekata tipa "`Razlomak`" treba podržati relacione operatore "`<`", "`>`", "`<=`", "`>=`","`==`" i "`!=`". Tu također treba biti oprezan kako to izvesti. Iz matemtike je poznato da ukoliko su p_1/q_1 i p_2/q_2 dva razlomka (pri čemu je $q_1 > 0$ i $q_2 > 0$), uvjet poput $p_1/q_1 \geq p_2/q_2$ sa matematičkog aspekta ekvivalentan je uvjetu $p_1 \cdot q_2 \geq p_2 \cdot q_1$, itd. Međutim, veliki je problem što pri računanju proizvoda $p_1 \cdot q_2$ i $p_2 \cdot q_1$ može doći do prekoračenja. Mada postoje i ovdje alternativne formule (recimo, može se koristiti ekvivalentni uvjet $(p_1/r) \cdot (q_2/s) \geq (p_2/r) \cdot (q_1/s)$ uz $r = \text{NZD}(p_1, p_2)$ i $s = \text{NZD}(q_1, q_2)$ kod kojeg je vjerovatnoća prekoračenja mnogo manja), za potrebe ovog zadatka sasvim dobro rješenje je izračunati *aproksimativne decimalne vrijednosti* razlomaka p_1/q_1 i p_2/q_2 (dijeljenjem brojnika sa nazivnikom) i uporediti dobijene decimalne vrijednosti. Pri tome, za dijeljenje treba koristiti tip "`long double`", jer samo on garantira dovoljnu preciznost da se ovo pouzdano obavi. Ipak, ovo rješenje nije univerzalno, jer na računarskim arhitekturama na kojima tip "`long long int`" ima više od 64 bita (tako da je moguće zapamtiti i brojeve sa više od 20 cifara), moguće je imati dva različita razlomka, ali čije su aproksimativne decimalne vrijednosti toliko bliske da se čak ni uz pomoć tipa "`long double`" ne može detektirati razlika.

Preklopljeni operator "<<" treba da podrži ispis objekata tipa "Razlomak" na ekran. Ukoliko je p brojnik a q nazivnik razlomka koji se ispisuje, ispis treba da izgleda kao p/q, osim ukoliko je nazivnik jednak jedinici, kada se ispisuje samo vrijednost brojnika. Slično, preklopljeni operator ">>" treba podrati unos objekata tipa "Razlomak" sa tastature. Potrebno je omogućiti da se ovi objekti unose u formi p/q, gdje su p i q cijeli brojevi, ali treba omogućiti da se unese i obični cijeli broj (u tom slučaju se podrazumijeva da je nazivnik jedinica). U slučaju neispravnog unosa, ulazni tok treba da dospije u neispravno stanje.

Konačno, treba podržati i automatsku konverziju objekata tipa "Razlomak" u tip "**long double**", koji omogućava upotrebu promjenljivih tipa "Razlomak" u kontekstima gdje se očekuju realni brojevi, tj. promjenljive tipa "**long double**" (treba znati da će ovim biti automatski podržana i konverzija u tip "**double**", pri čemu će ako se zahtijeva konverzija u tip "**double**", dodatne decimale koje nudi tip "**long double**" biti prosto odbačene). Rezultat konverzije je približna decimalna vrijednost koja se dobija dijeljenjem brojnika sa nazivnikom.

Sve metode obavezno implementirajte izvan deklaracije klase, osim trivijalnih metoda koje trebate implementirati direktno unutar deklaracije klase. Sve metode koje su inspektori, obavezno deklarirajte kao takve. Također, obavezno napišite i mali testni program u kojem čete testirati sve funkcionalnosti napisane klase.

2. Za lakše praćenje stanja dionica na nekoj berzi, potrebno je definirati i implementirati klasu nazvanu "Berza". Ova klasa omogućava čuvanje dnevnih informacija o tekućoj cijeni dionica za izvjesni vremenski period u vektoru cijelih brojeva, kojem se pristupa preko odgovarajućeg privatnog atributa (cijene se zadaju u feninzima, tako da se mogu iskazati kao cijeli broj). Interfejs klase sadrži sljedeće elemente:

   - Konstruktor sa dva parametra koji predstavljaju minimalnu i maksimalnu cijenu koja se može registrirati, te konstruktor sa jednim parametrom koji je maksimalna cijena koja se može registrirati (minimalna cijena se tada postavlja na nulu). Ovi parametri moraju biti pozitivni, u suprotnom treba baciti izuzetak tipa "range_error" uz prateći tekst "Ilegalne granicne cijene". Konstruktor sa jednim parametrom ne smije se koristiti za automatsku konverziju cjelobrojnih podataka u objekte tipa "Berza".
   - Metodu "RegistrirajCijenu" koja vrši registraciju nove cijene dionice zadane putem parametra. U slučaju da je cijena manja ili veća od minimalne odnosno maksimalne dozvoljene cijene, treba baciti izuzetak tipa "range_error" uz prateći tekst "Ilegalna cijena".
   - Metodu "DajBrojRegistriranihCijena" koja daje broj registriranih cijena.
   - Metodu "BrisiSve" koja briše sve unesene cijene.
   - Metode "DajMinimalnuCijenu" i "DajMaksimalnuCijenu" koje vraćaju kao rezultat minimalnu i maksimalnu registriranu cijenu (u slučaju da nema registriranih cijena, obje metode treba da bace izuzetak tipa "range_error" uz prateći tekst "Nema registriranih cijena"). Za realizaciju nije dozvoljeno koristiti petlje, nego isključivo odgovarajuće funkcije iz biblioteke "algorithm".
   - Unarni operator "!" koji primijenjen na objekat daje logičku vrijednost "tačno" ukoliko nema niti jedne registrirane cijene, u suprotnom daje logičku vrijednost "netačno".
   - Metodu "DajBrojCijenaVecihOd" koja vraća kao rezultat broj cijena većih od vrijednosti koja se zadaje kao parametar (u slučaju da nema registriranih cijena, metode treba da bace izuzetak tipa "range_error" uz prateći tekst "Nema registriranih cijena"). Za realizaciju nije dozvoljeno koristiti petlje, niti lambda funkcije ili pomoćne imenovane funkcije kriterija, nego isključivo samo odgovarajuće funkcije i/ili funktore iz biblioteka "algorithm" i "functional". Napomena: Ovdje ćete morati koristiti veznike, a na njihovu upotrebu treba se navići. Prvo probajte riješiti problem uz pomoć lambda funkcija. Kada Vam funkcija proradi, probajte istu funkcionalnost postići pomoću jednostavnijih ali zastarjelih veznika "bind1st" ili "bind2st" (koji su, usput, u verziji C++17 potpuno odbačeni), kao međukorak da bolje steknete osjećaj šta se zaista dešava. Kada Vam i to proradi, zamijenite zastarjele veznike sa boljim i univerzalnijim veznikom "bind" (zastarjele verzije veznika nemojte ostavljati u završnoj verziji).
   - Metodu "Ispisi" koja ispisuje sve unesene (registrirane) cijene sortirane u *opadajućem poretku* (tj. najveća cijena se ispisuje prva), pri čemu se svaka cijena ispisuje u posebnom redu. Cijene treba ispisivati u KM (ne u feninzima), pri čemu se uvijek prikazuju dvije decimale. Pri tome je neophodno koristiti funkcije i/ili funktore iz biblioteka "algorithm" i "functional". Upotreba

petlji, lambda funkcija ili pomoćnih imenovanih funkcija nije dozvoljena, ali je dozvoljeno koristiti i komponente iz nekih drugih biblioteka ukoliko smatrate da vam je to potrebno za realizaciju (od koristi će Vam biti i neke stvari rađene na Predavanju 7_b). Ova funkcija obavezno treba biti inspektor funkcija, tj. treba biti deklarirana s modifikatorom "`const`"!

- Preklopljeni operator "`[]`" koji omogućava da se direktno pročita *i*-ta registrirana cijena (numeracija ide od jedinice). Ukoliko je indeks izvan dozvoljenog opsega, treba baciti izuzetak tipa "`range_error`" uz prateći tekst "Neispravan indeks". Pri tome, ovaj operator se *ne može koristiti* za izmjenu podataka, odnosno ne može se koristiti sa lijeve strane znaka jednakosti.
- Preklopljene operatore "`++`" i "`--`" koji povećavaju odnosno smanjuju sve registrirane cijene za 1 KM. U slučaju da pri tome neka od cijena premaši maksimalno dozvoljenu vrijednost ili podbaci minimalno dozvoljenu vrijednost, baca se izuzetak tipa "`range_error`" uz prateći tekst "Prekoracen dozvoljeni opseg cijena". Potrebno je podržati kako prefiksnu, tako i postfiksnu verziju ovih operatora. Za realizaciju nije dozvoljeno koristiti petlje, niti lambda funkcije ili pomoćne imenovane funkcije, nego isključivo odgovarajuće funkcije i/ili funktore iz biblioteka "`algorithm`" i "`functional`" (vrijede iste napomene kao i ranije).
- Preklopljeni unarni operator "`-`" koji daje kao rezultat novi objekat tipa "`Berza`" u kojem su sve cijene oduzete od zbira maksimalno i minimalno dozvoljene cijene (ovim sve cijene i dalje sigurno ostaju u dozvoljenom opsegu). Za realizaciju nije dozvoljeno koristiti petlje, niti lambda funkcije ili pomoćne imenovane funkcije, nego isključivo odgovarajuće funkcije i/ili funktore iz biblioteka "`algorithm`" i "`functional`" (vrijede iste napomene kao i ranije).
- Preklopljene operatore "`+`" i "`-`" koji djeluju na sljedeći način. Ukoliko je "`X`" objekat tipa "`Berza`", a "`Y`" cijeli broj, tada je "`X + Y`" je novi objekat tipa "`Berza`" u kojem su sve registrirane cijene povećane za iznos "`Y`". Slično, "`X - Y`" je novi objekat tipa "`Berza`" u kojem su sve registrirane cijene umanjene za iznos "`Y`". Pri istim tipovima operanada "`X`" i "`Y`", izraz "`Y + X`" treba da ima isto značenje kao i izraz "`X + Y`", dok izraz "`Y - X`" predstavlja novi objekat u kojem su sve registrirane cijene oduzete od vrijednosti "`Y`". Ukoliko se kao rezultat ovih operacija dogodi da neka od cijena izađe izvan dozvoljenog opsega, treba baciti izuzetak tipa "`domain_error`" uz prateći tekst "Prekoracen dozvoljeni opseg cijena". U slučaju kada su i "`X`" i "`Y`" objekti tipa "`Berza`", tada je izraz "`X - Y`" novi objekat tipa "`Berza`" koji sadrži *razlike* odgovarajućih cijena iz objekata "`X`" i "`Y`". U ovom posljednjem slučaju se podrazumijeva da "`X`" i "`Y`" sadrže isti broj registriranih cijena, te da imaju iste minimalne i maksimalne dozvoljene cijene (u suprotnom, treba baciti izuzetak tipa "`domain_error`" uz prateći tekst "Nesaglasni operandi"). Ukoliko pri tome bilo koja od razlika izađe izvan dozvoljenog opsega cijena, treba baciti isti izuzetak kao i u prethodnim slučajevima. U svim ostalim situacijama, izrazi "`X + Y`" odnosno "`X - Y`" ne trebaju biti sintaksno ispravni. Za realizaciju koristiti isključivo prikladne funkcije i/ili funktore iz biblioteka "`algorithm`" i "`functional`" (vrijede iste napomene kao i ranije).
- Preklopljene operatore "`+=`" i "`-=`" čiji je cilj da značenje izraza oblika "`X += Y`" odnosno "`X -= Y`" bude identično značenju izraza "`X = X + Y`" i "`X = X - Y`" kad god oni imaju smisla.
- Preklopljene relacione operatore "`==`" i "`!=`" koje ispituju da li su dva objekta tipa "`Berza`" jednaka ili nisu. Dva objekta ovog tipa smatraju se jednakim ukoliko sadrže isti broj registriranih cijena, i ukoliko su sve odgovarajuće registrirane cijene u oba objekta jednake. Za realizaciju ovih operatora koristite isključivo odgovarajuće funkcije i/ili funktore iz biblioteka "`algorithm`" i "`functional`" (vrijede iste napomene kao i ranije).

Sve metode implementirajte izvan klase, osim metoda čija je implementacija vrlo kratka. Metode koje su po prirodi inspektori obavezno treba deklarirati kao takve. Obavezno napišite i mali testni program u kojem će se testirati sve elemente napisane klase

3. Za vođenje evidencije podataka o robi u nekom skladištu potrebno je razviti kontejnersku klasu nazvanu "`Skladiste`". U skladištu se roba nalazi pohranjena u sanducima (za čvrste predmete), u vrećama (za praškaste materije) i u buradima (za tečnosti). Sanduci, vreće i burad se modeliraju redom pomoću klasa "`Sanduk`", "`Vreca`" odnosno "`Bure`". Sanduk je opisan svojom težinom u kilogramima (realni broj), nazivom predmeta koji se u njemu čuvaju (tipa "`string`"), pri čemu se pretpostavlja se da jedan sanduk čuva samo istovrsne predmete), te težinama predmeta koji se u njemu čuvaju, također u kilogramima (ovi podaci se čuvaju u vektoru realnih brojeva, pri čemu svaki od elemenata vektora odgovara jednom predmetu). Vreća je također opisana svojom težinom (u kilogramima), nazivom praškaste materije koji se u njoj čuva, te težinom pohranjene praškaste materije. Bure je opisano svojom težinom, nazivom tečnosti koja se u njemu čuva, specifičnom

težinom (gustinom) tečnosti u kilogramima po metru kubnom (realan broj), te zapreminom pohranjene tečnosti koja se u njemu čuva. Informacijama o robi pohranjenoj u skladištu pristupa se pomoću vektora pametnih pokazivača koji pokazuju na objekte tipa "Sanduk", "Vreca" ili "Bure" (za tu svrhu, sve ove klase moraju biti izvedene iz neke apstraktne bazne klase, koju ćete nazvati "Spremnik"). Tom nizu pokazivača se pristupa preko nekog od atributa pohranjenog unutar klase "Skladiste". Konstruktor klase "Sanduk" kao parametre zahtijeva težinu, naziv predmeta koji se čuvaju, te vektor težina predmeta koji se u njemu čuvaju. Konstruktor klase "Vreca" prima kao parametre težinu vreće, naziv pohranjene materije, te težinu pohranjene materije, dok konstruktor klase "Bure" prima kao parametre težinu, naziv tečnosti koja se čuva, specifičnu težinu tečnosti, te zapreminu pohranjene tečnosti. Atribute koji su zajednički za sanduke, vreće i burad (težina i naziv pohranjenog sadržaja) treba čuvati u baznoj klasi, dok izvedene klase čuvaju samo atribute specifične za datu vrstu objekta. U baznoj klasi također trebaju biti i metode "DajTezinu", "DajUkupnuTezinu" i "Ispisi" bez parmetara. Prva metoda daje težinu vlastitu težinu sanduka, vreće ili bureta (bez onoga što je u njima), druga radi istu stvar, samo uračunava u obzir i težinu onoga što se nalazi u sanduku ili buretu, dok metoda "Ispisi" ispisuje podatke o sanduku ili buretu u obliku poput sljedećeg:

```
Vrsta spremnika: Sanduk
Sadrzaj: Trofazni kataklingeri za auspuhe
Tezine predmeta: 2 3 1 2 2 4 3 1 3 (kg)
Vlastita tezina: 10 (kg)
Ukupna tezina: 31 (kg)

Vrsta spremnika: Vreca
Sadrzaj: Praskaste cincozne za glajfanje
Vlastita tezina: 0.2 (kg)
Tezina pohranjene materije: 5 (kg)
Ukupna tezina: 5.2 (kg)

Vrsta spremnika: Bure
Sadrzaj: Rafinirana kalamuta iz Katange
Vlastita tezina: 5 (kg)
Specificna tezina tecnosti: 1300 (kg/m^3)
Zapremina tecnosti: 150 (l)
Ukupna tezina: 200 (kg)
```

Razumije se da će metode "DajUkupnuTezinu" i "Ispisi" morati biti apstraktne, s obzirom da u baznoj klasi nema dovoljno informacija koje omogućavaju njihovu implementaciju

Što se tiče klase "Skladiste", primjerci ove klase treba da se mogu kreirati ne navodeći nikakve dodatne specifikacije. Pri tome, potrebno je podržati da se prilikom kopiranja ili međusobnog dodjeljivanja primjeraka ove klase kreiraju potpune (duboke) a ne plitke kopije. Ovo kopiranje mora da korektno radi i ukoliko se u budućnosti pojavi neki novi tip spremnika osim sanduka, vreća i buradi (s obzirom da se radi o polimorfnoj kolekciji objekata, za tu svrhu će biti potrebno dodati odgovarajuću podršku i u ostale klase).

Za manipulaciju sa podacima u skladištu predviđeno je nekoliko metoda. Na prvom mjestu, metode "DodajSanduk", "DodajVrecu" odnosno "DodajBure" kreiraju novi objekat tipa "Sanduk", "Vreca" odnosno "Bure" i pohranjuju ga u skladište. Parametri ovih metoda isti su kao i parametri konstruktora objekata tipa "Sanduk", "Vreca" odnosno "Bure". Sve ove tri metode kao rezultat vraćaju adresu novokreiranog objekta u formi (običnog) pokazivača čiji je statički tip pokazivač na "Spremnik" (ova povratna vrijednost će se obično ignorirati, ali vidjećemo uskoro gdje ova povratna vrijednost može biti bitna). Predviđena je i univerznalna metoda za dodavanje sa dva parametra nazvana "DodajSpremnik", koja može dodavati objekte bilo kojeg tipa izvedenog iz bazne klase "Spremnik". Prvi parametar je pokazivač na objekat koji želimo dodati, dok je drugi parametar logička vrijednost koja određuje da li se objekat predaje u vlasništvo klasi ili ne. Ukoliko ovaj parametar ima vrijednost "**true**", pretpostavlja se da je onaj ko poziva ovu metodu kreirao dinamički ovaj objekat i ne želi se više brinuti o njemu, nego brigu o njemu prepušta kontejnerskoj klasi (tačnije, pametnim pokazivačima koji se u njoj nalaze). S druge strane, ukoliko ovaj parametar ima vrijednost "**false**", objekat čiju adresu šaljemo u funkciju ne mora uopće biti dinamički alociran, a i ukoliko jeste, o njegovom brisanju se treba brinuti onaj ko je pozvao funkciju. U tom slučaju (s obzirom da kolekcija svakako mora čuvati dinamički alocirane objekte), treba kreirati

dinamičku kopiju tog objekta koja se pohranjuje u kolekciju umjesto samog objekta. Funkcija kao rezultat vraća adresu pohranjenog objekta, što ovisno o slučaju može biti ista adresa koja je poslana kao parametar, ili adresa kreirane dinamičke kopije. Metoda "`BrisiSpremnik`" omogućava brisanje nekog od objekata koji su već pohranjeni u skladištu. Parametar ove funkcije je adresa objekta koji se briše, i to je zapravo ista adresa koju vraćaju funkcije poput "`DodajSanduk`". Slijedi da ako želimo imati mogućnost da brišemo neki objekat (npr. sanduk), moramo negdje sačuvati adresu koja je vraćena prilikom kreiranja objekta, jer je taj objekat moguće izbrisati jedino navodeći tu adresu (s obzirom da ne postoji nikakav drugi identifikator koji bi mogao jednoznačno identificirati taj objekat). Metode "`DajNajlaksi`" odnosno "`DajNajtezi`" vraćaju reference na najlakši odnosno najteži objekat (sanduk, vreća ili bure) u skladištu, ne računajući ono što je pohranjeno u tom objektu. Ove metode se ne smiju moći pozvati nad konstantnim objektima tipa "`Skladiste`". U slučaju da je skladište prazno, treba baciti izuzetak tipa "`range_error`" uz prateći tekst "Skladiste je prazno". Metoda "`BrojPreteskih`" vraća broj objekata u skladištu čija je ukupna težina (tj. vlastita težina zajedno sa težinom onoga što se u njima nalazi) veća od iznosa koji se zadaje putem cjelobrojnog parametra. Ova metoda se mora moći pozvati i nad konstantnim objektom. Predviđena je i metoda "`IzlistajSkladiste`" koja ispisuje spisak svega što se nalazi u skladištu, sortiran u opadajući poredak po ukupnoj težini. Ispis se vrši pozivom metode "`Ispisi`", bez ikakvih praznih redova između objekata. Metoda se mora moći pozvati i nad konstantnim objektom. Konačno, treba implementirati i metodu "`UcitajIzDatoteke`" koja čita podatke o robi iz tekstualne datoteke čije se ime zadaje kao parametar i smješta robu u skladište (u slučaju da u skladištu već ima unesene robe, postojeći podaci se brišu). Svaki objekat opisan je sa dva reda u datoteci. U prvom redu se nalazi početno slovo "`S`", "`V`" ili "`B`" (za sanduk, vreću odnosno za bure) iza kojeg nakon jednog razmaka slijedi naziv predmeta, praškaste materije ili tečnosti koje su pohranjene u sanduku, vreći odnosno buretu (npr. "`S Tepsije`", "`V Brasno`" ili "`B Suncokretovo ulje`"). U drugom redu se za slučaj sanduka nalazi težina sanduka, broj predmeta i težina svakog od njih, razdvojeno po jednim razmakom (npr. "`10 9 2 3 1 2 2 4 3 1 3`"), za slučaj vreće tu su težina vreće i težina pohranjene materije (npr. "`0.2 5`") dok se za slučaj bureta nalazi težina bureta, te specifična težina i zapremina tečnosti (npr. "`5 1300 150`"). Ukoliko tražena datoteka ne postoji, treba baciti izuzetak tipa "`logic_error`" uz prateći tekst "Trazena datoteka ne postoji". Isti izuzetak, ali uz prateći tekst "Datoteka sadrzi besmislene podatke" treba baciti ukoliko podaci u datoteci nisu u skladu sa specifikacijama. U slučaju bilo kakvih drugih problema pri čitanju (osim pokušaja čitanja iza kraja datoteke), treba također baciti isti izuzetak, uz prateći tekst "Problemi pri citanju datoteke".

Razvijene klase demonstrirajte u testnom programu koji iščitava podatke iz tekstualne datoteke sa imenom "`ROBA.TXT`", a nakon toga ispisuje spisak svega što se nalazi u skladištu, sortiran u opadajući poredak po ukupnoj težini. U testnom programu obavezno predvidite i hvatanje eventualno bačenih izuzetaka koji se mogu pojaviti.

4. Implementirajte surogatsku klasu "`PolimorfniSpremnik`" koja predstavlja polimorfni omotač za proizvoljnu vrstu spremnika iz prethodnog zadatka. Preciznije, promjenljive ovog tipa moraju biti takve da se u njih može smjestiti bilo sanduk, bilo vreća, bilo bure (tj. sadržaj promjenjive čiji je tip bilo tip "`Sanduk`", bilo tip "`Vreca`", bilo tip "`Bure`"), pa čak i promjenljiva bilo kojeg tipa koji je izveden iz apstraktnog tipa "`Spremnik`" (što uključuje i tipove koji će eventualno biti kreirani u budućnosti). Naravno, sa promjenljivim tipa "`PolimorfniSpremnik`" mogu se raditi sve operacije koje su predviđene da rade sa svim vrstama spremnika neovisno od njihovog podtipa (tj. sve operacije koje su definirane u interfejsu apstraktne bazne klase "`Spremnik`"), mogu se bezbjedno kopirati, međusobno dodjeljivati, itd. Na primjer:

```
PolimorfniSpremnik s1 = Bure(5,"Benzin", 930, 70);        // s1 je bure
PolimorfniSpremnik s2, s3;                                // s2 i s3 su nespecificirani
s2 = Sanduk(3, "Tepsije", {0.5, 0.8, 0.6, 0.5});          // s2 je sada sanduk
s3 = Vreca(0.4, "Brasno", 30);            // a s3 vreca
std::cout << s1.DajTezinu() << std::endl;
std::cout << s2.DajUkupnuTezinu() << std::endl;
s3.Ispisi();
s1 = s2;                                                  // sad je i s1 sanduk...
s1.Ispisi();
```

Ukoliko se kreira objekat tipa "`PolimorfniSpremnik`" bez ikakvih dodatnih informacija, on je na početku nespecificiran i svaki pokušaj poziva bilo koje od metoda nad takvim objektom treba

baciti izuzetak tipa "logic_error" uz prateći tekst "Nespecificiran spremnik". Obavezno napišite i testni program u kojem ćete testirati funkcionalnost razvijene surogatske klase.

5. Dopunite generičku klasu "Matrica" razvijenu u Zadatku 6 iz Riješenih zadataka uz Predavanje 12 s četiri nove metode nazvane redom "SacuvajUTekstualnuDatoteku", "SacuvajUBinarnuDatoteku", "ObnoviIzTekstualneDatoteke" i "ObnoviIzBinarneDatoteke", te jednim dodatnim konstruktorom, koji će kasnije biti opisan. Sve ove metode primaju kao parametar naziv datoteke (tipa "string"). Metoda "SacuvajUTekstualnuDatoteku" snima sadržaj matrice nad kojom je pozvana u tekstualnu datoteku čije je ime zadano. Datoteka treba formatirana tako da se podaci o svakom redu matrice čuvaju u posebnim redovima datoteke, pri čemu su elementi unutar jednog reda međusobno razdvojeni zarezima (iza posljednjeg reda nema zareza). Recimo, za neku matricu formata 3×4 kreirana datoteka može izgledati poput sljedeće:

```
2.5,-3,1.12,4
0,0.25,3.16,42.3
-1.7,2.5,0,5
```

U slučaju da dođe do bilo kakvih problema pri upisu, treba baciti izuzetak tipa "logic_error" uz prateći tekst "Problemi sa upisom u datoteku". Metoda "SacuvajUBinarnuDatoteku" obavlja sličnu funkcionalnost, samo što se upis vrši u binarnu datoteku, u koju je potrebno snimiti one podatke iz memorije koji su neophodni da bi se kasnije mogla pouzdano izvršiti rekonstrukcija stanja matrice iz pohranjenih podataka. Pri tome se podrazumijeva da je tip elemenata matrice neki skoro-POD tip podataka (u suprotnom, snimanje u binarnu datoteku najvjerovatnije neće biti korektno). U slučaju problema pri upisu, vrijedi isto kao kod prethodne metode. Dalje, metode "ObnoviIzTekstualneDatoteke" odnosno "ObnoviIzBinarneDatoteke" vrše obnavljanje sadržaja matrice na osnovu sačuvanog stanja u tekstualnoj odnosno binarnoj datoteci, pri čemu se prethodni sadržaj matrice uništava. U oba slučaja, ukoliko tražena datoteka ne postoji, treba baciti izuzetak tipa "logic_error" uz prateći tekst "Trazena datoteka ne postoji". Pri čitanju iz tekstualne datoteke, ukoliko datoteka sadrži podatke koji nisu u skladu sa tipom elemenata matrice, ukoliko podaci nisu razdvojeni zarezima, ili ukoliko različiti redovi imaju različit broj elemenata, treba baciti isti izuzetak, uz prateći tekst "Datoteka sadrzi besmislene podatke" (s obzirom da u tekstualnoj datoteci nije pohranjena nikakva informacija o broju redova i kolona, datoteku ćete morati efektivno iščitati dva puta, prvi put da saznate broj redova i kolona, a drugi put da zaista pročitate vrijednosti elemenata, nakon što su obavljene odgovarajuće dinamičke alokacije). U slučaju bilo kakvih drugih problema pri čitanju, treba također baciti isti izuzetak, uz prateći tekst "Problemi pri citanju datoteke". Za slučaj čitanja iz binarne datoteke, u slučaju bilo kakvih problema (osim nepostojeće datoteke), treba baciti ovaj isti izuzetak.

Konačno, rečeno je da je u klasu "Matrica" potrebno dodati i novi konstruktor. Taj konstruktor će imati dva parametra, pri čemu je prvi parametar ime datoteke, a drugi je logička vrijednost koja određuje da li će se konstrukcija objekta vršiti iz tekstualne datoteke (u slučaju kad taj parametar ima vrijednost "**false**") ili iz binarne datoteke (kada parametar ima vrijednost "**true**"). Ovaj konstruktor ponaša se identično kao što se ponašaju metode "ObnoviIzTekstualneDatoteke" odnosno "ObnoviIzBinarneDatoteke", samo se oslanja na činjenicu da se objekat tek stvara, tako da nema potrebe za oslobađanjem resursa koje je objekat prije toga koristio.

Obavezno napišite i kratki testni program u kojem ćete testirati novododane funkcionalnosti klase vezane za datoteke. Ostale funkcionalnosti klase ne morate testirati, s obzirom da se podrazumijeva da ste ih testirali ranije. Naravno, dozvoljena je upotreba i ostalih funkcionalnosti ukoliko su Vam potrebne za potrebe testiranje onoga što se traži da testirate (recimo, da formirate matrice koje želite snimiti, ili da ispišete sadržaj obnovljene matrice).

6. Potrebno je napraviti generičku funkciju "SortirajBinarnuDatoteku" koja omogućava sortiranje podataka pohranjenih u binarnoj datoteci *bez učitavanja sadržaja datoteke u memoriju*. Funkcija treba da ima sljedeći prototip:

```
template <typename TipElemenata>
  void SortirajBinarnuDatoteku(const char ime_datoteke[],
    std::function<bool(TipElemenata, TipElemenata)> kriterij
      = std::less<TipElemenata>());
```

Prvi parametar je ime datoteke, za koju se podrazumijeva da je organizirana kao kolekcija elemenata istog tipa "TipElemenata" (koji naravno mora biti neki skoro-POD tip), snimljenih u

datoteku jedan za drugim. Pri tome je posve nebitno kako je ta kolekcija nastala (da li snimanjem elemenata jedan za drugim, "istresanjem" čitavog niza elemenata odjednom, ili na neki treći način). Drugi parametar je funkcija kriterija, koja radi na posve analogan način kao funkcija kriterija u bibliotečkoj funkciji "`sort`", pri čemu se može zadati bilo obična funkcija, bilo lambda funkcija, bilo funkcijski objekat (funktor), tačnije bilo šta što se može pozvati sa dva argumenta koji su tipa "`TipElemenata`" (ili se mogu konvertovati u taj tip) i koja vraća logičku vrijednost (ili nešto što se može interpretirati kao logička vrijednost). Recimo, ukoliko datoteka "`BROJEVI.DAT`" sadrži cijele brojeve, njen sadržaj možemo sortirati u opadajući poredak pozivom poput

```
SortirajBinarnuDatoteku<int>("BROJEVI.DAT", [](int x, int y) { return x > y; });
```

Nažalost, tip elemenata se mora eksplicitno specificirati (tj. nije moguće pisati samo nešto poput "`SortirajBinarnuDatoteku( … )`" umjesto "`SortirajBinarnuDatoteku<int>( … )`"), odnosno tip se ne može deducirati iz parametara lambda funkcije, jer je dedukcija moguća samo u slučaju potpunog slaganja tipa (drugi parametar je deklariran kao polimorfni funkcijski omotač, kojem se zaista može dodijeliti lambda funkcija, ali lambda funkcija nije polimorfni funkcijski omotač, pa nemamo potpuno slaganje po tipu). Koristeći funkcijske objekte iz bibiloteke "`functional`", prethodni poziv mogli smo kraće izvesti kao

```
SortirajBinarnuDatoteku<int>("BROJEVI.DAT", std::greater<int>());
```

Drugi parametar ima podrazumijevanu vrijednost "`std::less<TipElemenata>()`", tako da se može izostaviti ukoliko želimo sortiranje u podrazumijevani rastući poredak.

U slučaju da zadana datoteka ne postoji, treba baciti izuzetak tipa "`logic_error`" uz prateći tekst "Datoteka ne postoji". U slučaju bilo kakvih drugih problema pri čitanju ili pisanju datoteke treba baciti isti izuzetak uz prateći tekst "Problemi u pristupu datoteci".

Napisanu funkciju iskoristite u testnom programu koji će testirati rad ove funkcije. Taj testni program moraće prvo kreirati neku binarnu datoteku, zatim pozvati ovu funkciju, i onda konačno iščitati njen sadržaj, da se uvjerimo da je datoteka zaista sortirana. Najstrože je zabranjeno da funkcija "`SortirajBinarnuDatoteku`" učita sadržaj datoteke u memoriju, sortira ga u memoriji i zatim vrati sortirani sadržaj nazad. Sortiranje se mora obaviti isključivo manipuliranjem nad sadržajem datoteke. Kršenje ove zabrane rezultiraće *dodjelom nula poena na ovaj zadatak*. Za sortiranje koristite neki od naivnih algoritama sortiranja koji Vam je poznat. Vodite računa da, uz pogodne funkcije kriterija, ova funkcija mora biti u stanju da sortira obje datoteke "`BROJEVI.DAT`" i "`STUDENTI.DAT`" koje su demonstrirane na Predavanju 14_b!