Zadaća 2.

Ova zadaća nosi ukupno 4,5 poena, pri čemu prvi zadatak nosi 1 poen, drugi zadatak 1,5 poen, a treći i četvti zadatak po 1 poen. Svi zadaci se mogu uraditi na osnovu gradiva sa prvih 6 predavanja i pretpostavljenog predznanja iz predmeta "Osnove računarstva". Rok za predaju ove zadaće je ponedjeljak, 24. IV 2017. do 9.00. Napomena: svi zadaci su mnogo kraći nego njihov opis!

1. Minesweeper je poznata igra za jednog igrača, pri čemu se igraču nudi tabla (ploča) formata $n \times n$ polja na kojima je raspoređeno m mina. Tabla je opisana vektorom vektôrā čiji su elementi tipa "`Polje`", koji predstavlja pobrojani tip, definiran na globalnom nivou deklaracijom

```
enum class Polje {
  Prazno, Posjeceno, Mina, BlokiranoPrazno, BlokiranoPosjeceno, BlokiranoMina
};
```

Također, na globalnom nivou postoji deklaracija

```
typedef std::vector<std::vector<Polje>> Tabla;
```

Cilj igre je otkriti sva polja na tabli koja nemaju mine ne detonirajući (otkrivajući) pri tome niti jednu minu. Implementirajte sljedeće funkcije koje simuliraju jednu partiju minesweepera:

```
Tabla KreirajIgru(int n, const std::vector<std::vector<int>> &mine);
std::vector<std::vector<int>> PrikaziOkolinu(const Tabla &polja, int x, int y);
void BlokirajPolje(Tabla &polja, int x, int y);
void DeblokirajPolje(Tabla &polja, int x, int y);
Status Idi(Tabla &polja, int &x, int &y, Smjerovi smjer);
Status Idi(Tabla &polja, int &x, int &y, int novi_x, int novi_y);
std::vector<std::vector<int>> PrikaziOkolinu(const Tabla &polja, int x, int y);
```

Funkcija "`KreirajIgru`" prima kao prvi parametar cijeli broj koji određuje veličinu igraće table, te je na osnovu ovog parametra potrebno kreirati tablu koja će incijalno sadržavati sva polja inicijalizirana na vrijednost "`Prazno`". Drugi parametar funkcije je vektor vektôrā koji označava polja na kojima se nalaze mine. Svaki od vektôrā u ovom vektoru vektôrā ima dva elementa, od kojih prvi definira horizontalnu, a drugi vertikalnu poziciju mine (numeracija kreće od 0). Tako, ukoliko je vektor vektôrā "`mine`" jednak {{0,0}, {0,2}, {1,3}}, tada se mine nalaze na prvom lijevom gornjem polju, na trećem polju u prvom redu te na četvrtom polju u drugom redu. U matrici koja će predstavljati igraću tablu a koja je kreirana u ovoj funkciji, na mjestu svake mine potrebno je postaviti vrijednost na "`Mina`". Tako bi za slučaj veličine igraće table od 4×4 i prethodno navedenog primjera matrice mina, matrica koja opisuje igru izgledala ovako, pri čemu je "`Prazno`" prikazano kao 0, a "`Mina`" kao 1:

1 0 1 0
0 0 0 1
0 0 0 0
0 0 0 0

Funkcija treba vratiti kreiranu igraću tablu. Ukoliko svaki od vekôrā unutar parametra "`mine`" ne sadrži tačno dva elementa, treba baciti izuzetak tipa "`domain_error`" uz prateći tekst "Ilegalan format zadavanja mina". Ukoliko neka od zadanih pozicija mina izlazi izvan opsega table, treba baciti izuzetak istog tipa, ali uz prateći tekst "Ilegalne pozicije mina".

Funkcija "`PrikaziOkolinu`" treba da vrati matricu formata 3×3 koja za sva polja koja se nalaze u neposrednom susjedstvu polja (x, y) čije se koordinate zadaju putem parametara, uključujući i sâmo polje (x, y), daje broj mina koje se nalaze u neposrednom susjedstvu tih polja (ne računajući eventualnu minu na razmatranom polju). Na primjer, uzmimo da je $x = 1$ i $y = 1$, a da isječak iz table izgleda ovako, uz istu konvenciju o prikazu kao i maloprije:

0 1 0 0 1 …
0 0 0 1 1 …
1 1 0 1 0 …
0 0 0 0 0 …

Funkcija tada treba da vrati matricu $\{\{1,0,2\},\{3,3,4\},\{1,1,3\}\}$, što predstavlja broj mina u neposrednoj okolini pôljā (0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1) i (2, 2) respektivno. Van table se smatra da nema mina (ovo je bitno ako se testira susjedstvo nekog polja koje se nalazi tačno na rubu table). Ukoliko se polje nalazi izvan table, funkcija treba baciti izuzetak tipa "`domain_error`" uz prateći tekst "Polje (x, y) ne postoji", pri čemu x odnosno y predstavljaju koordinate polja.

Funkcija "`BlokirajPolje`" označava polje čije se koordinate prosljeđuju putem parametara "`x`" i "`y`" kao neaktivno, tako da se igrač na to polje više ne može pomjerati. Funkcija treba u tabli (tj. matrici igre koja se prosljeđuje kao prvi parametar) na poziciju ovog polja upisati vrijednost "`BlokiranoPrazno`", "`BlokiranoPosjeceno`" ili "`BlokiranoMina`" ovisno od toga da li se na tom polju ranije nalazila vrijednost "`Prazno`", "`Posjeceno`" ili "`Mina`". U slučaju da se pokuša blokirati polje koje ne postoji (izvan igraće table), funkcija treba vratiti isti izuzetak kao u prethodnoj funkciji. U slučaju da se pokuša blokirati polje koje je već blokirano funkcija ne treba uraditi ništa.

Funkcija "`DeblokirajPolje`" radi operaciju obrnutu od one u funkciji "`BlokirajPolje`", pri čemu se vrijednost na polju koje se deblokira treba vratiti na vrijednost koja je bila na tom polju prije blokiranja. Ako se pokuša deblokirati polje koje ne postoji, funkcija treba baciti isti izuzetak kao u prethodne dvije funkcije. U slučaju da se pokušava deblokirati polje koje nije blokirano, funkcija ne treba uraditi ništa.

Funkcija "`Idi`" ima dvije verzije, sa četiri i sa pet parametara. Obje verzije primaju reference na tablu, te reference na varijable koje čuvaju trenutnu poziciju igrača. Verzija sa 4 parametra prima još i smjer u kom se igrač želi kretati. Taj parametar je tipa "`Smjerovi`" koji je pobrojani tip deklariran na globalnom nivou kao

```
enum class Smjerovi {
  GoreLijevo, Gore, GoreDesno, Desno, DoljeDesno, Dolje, DoljeLijevo, Lijevo
};
```

Ova verzija funkcije treba na osnovu trenutne pozicije igrača te smjera koji je poslan funkciji "pomjeriti" igrača za jedno polje u datom smjeru, tako što će ažurirati vrijednost pozicije igrača, a polje na kojem se igrač nalazio prije toga označava se kao posjećeno (vrijednost "`Posjeceno`"). Posjećena polja se po svemu tretiraju kao i prazna polja, osim po jednom detalju koji će uskoro biti objašnjen. Za razliku od ove verzije, verzija sa 5 parametara direktno prima kao parametre koordinate novog polja na koji se želimo pomjeriti. U slučaju da igrač pokušava da se pomjeri na polje koje ne postoji, treba baciti izuzetak tipa "`out_of_range`" uz prateći tekst "Izlazak van igrace table". Ukoliko se pokušamo pomjeriti na mjesto koje je označeno kao blokirano, treba baciti izuzetak tipa "`logic_error`" uz prateći tekst "Blokirano polje". Funkcija kao rezultat vraća vrijednost tipa "`Status`", koji je pobrojani tip definiran na globalnom nivou kao

```
enum class Status {
  NijeKraj, KrajPoraz, KrajPobjeda
};
```

U slučaju da igrač stane na polje na kome se nalazi mina, funkcija treba vratiti "`KrajPoraz`", te očistiti matricu koja označava polja (postaviti vrijednost svih polja na "`Prazno`"). U slučaju da je igrač posjetio sva polja na kojima nema mina, odnosno nema više praznih polja (radi ovoga je potrebno označavati posjećena polja), funkcija treba vratiti vrijednost "`KrajPobjeda`". U svim ostalim slučajevima funkcija vraća vrijednost "`NijeKraj`".

Za potrebe komunikacije između programa i korisnika, potrebno je implementirati još neke upravljačke funkcije. Funkcija "`PrijaviGresku`" ima parametar tipa "`KodoviGresaka`" koji predstavlja pobrojani tip definiran na globalnom nivou kao

```
enum class KodoviGresaka {
  PogresnaKomanda, NedostajeParmetar, SuvisanParametar, NeispravanParametar
};
```

Ova funkcija, u zavisnosti od vrijednosti parametra koji joj je proslijeđen, na ekran ispisuje neki od tekstova koji se mogu javiti kao greška u radu u toku igre, u skladu sa sljedećom tabelom:

Kôd greške:	Tekst koji treba ispisati:
`PogresnaKomanda`	Nerazumljiva komanda!
`NedostajeParametar`	Komanda trazi parametar koji nije naveden!
`NeispravanParametar`	Parametar komande nije ispravan!
`SuvisanParametar`	Zadan je suvisan parametar nakon komande!

Korisniku programa treba omogućiti unos komandi kroz konzolni prozor koji će implementirati funkcija

```
bool UnosKomande(Komande &komanda, Smjerovi &smjer, int &x, int &y,
  KodoviGresaka &greska);
```

gdje je "`Komande`" pobrojani tip definisan kao

```
enum class Komande {
  PomjeriJednoMjesto, PomjeriDalje, Blokiraj, Deblokiraj, PrikaziOkolinu,
  ZavrsiIgru, KreirajIgru
};
```

Ova funkcija očekuje od korisnika da zada komandu putem tastature. Legalne komande su sljedeće ("`1`" u "`P1`", te "`>`" u "`P>`" je dio komande, a ne parametar):

Komanda:	Značenje:
`P1` *smjer*	Pomjeri se za jedno mjesto
`P>` x y	Pomjeri se na zadano polje
`B` x y	Blokiraj polje
`D` x y	Deblokiraj polje
`PO`	Prikaži okolinu trenutnog polja
`Z`	Završi igru
`K`	Kreiraj igru

Razmaci ispred i iza komande su dozvoljeni (mada ne i obavezni), ali bilo kakav neočekivani znak (osim razmaka) nakon komande tretira se kao *suvišan parametar*. U slučaju da se prepozna ispravna komanda, funkcija smješta njen kôd u parametar "`komanda`", dok se njeni eventualni parametari komande smještaju se sljedeća tri parametra, zavisno od konkretne komande.

Komanda "`P1`" je praćena jednim od znakova/kombinacija znakova "`GL`", "`G`", "`GD`", "`D`", "`DoD`", "`Do`", "`DoL`" ili "`L`", koji respektivno predstavljaju smjerove "`GoreLijevo`", "`Gore`", "`GoreDesno`", "`Desno`", "`DoljeDesno`", "`Dolje`", "`DoljeLijevo`" odnosno "`Lijevo`" a definiraju u kom pravcu je potrebno pomjeriti robota-igrača (za jedno mjesto). Razmaci između komande i parametra su dozvoljeni, kao i razmaci iza parametra, ali bilo šta što ne predstavlja ispravan parametar sa prethodne liste (uključujući suvišne znakove iza) tretiraju se kao neispravan parametar. Smjer treba upisati u parametar "`smjer`", dok ostali parametri funkcije ostaju nedefinirani.

Komadne "`P>`", "`B`" i "`D`" su praćene sa dva cijela broja, koji predstavljaju polje na koje igrač treba da se premjesti nakon izvršenja ove komande (za komandu "`P>`"), odnosno polje koje treba blokirati/deblokirati za komande "`B`"/"`D`". Razmaci između komande i parametra su dozvoljeni, ali nikakvi znakovi između dva parametra ili iza drugog parametra (osim razmaka) nisu dozvoljeni i tretiraju se kao neispravan parametar. U slučaju korektnog unosa, u parametre "`x`" i "`y`" upisuju se vrijednosti parametara unesenih iza komande. Komande "`Z`", "`PO`" i "`K`" se unose bez parametara (razmaci prije i nakon komande su dozvoljeni).

U slučaju da se komanda ne može prepoznati, odnosno nije unesena u nekom od prethodno prikazanih formata, potrebno je u parametar "`kod_greske`" upisati vrijednost "`PogresnaKomanda`". U slučaju unosa komande koja očekuje parametar, a taj parametar nije obezbijeđen pri unosu, u parametar "`kod_greske`" treba upisati vrijednost "`NedostajeParametar`", dok ukoliko se pronađe nešto što nije razmak iza komande koja ne zahtijeva parametar, ili nešto što nije razmak iza posljednjeg parametra komande koja zahtijeva parametar, tada kao kôd greške treba upisati vrijednostu "`SuvisanParametar`".

Ako je komanda unesena ispravno, funkcija vraća kao rezultat logičku vrijednost "`true`" kao signal da je komanda prepoznata. Parametar "`kod_greske`" tada je nedefiniran. U suprotnom, ukoliko nije prepoznata ispravna komanda, kôd greške se smješta u parametar "`kod_greske`", ostali parametri su nedefinirani, a funkcija vraća kao rezultat logičku vrijednost "`false`" kao signal da nije prepoznata ispravna komanda. Potrebno je predvidjeti sve što bi korisnik eventualno mogao unijeti (funkcija kao i program koji je koristi ne smije da "crkne" šta god korisnik unio).

Naposlijetku, potrebno je napisati funkciju

```
void IzvrsiKomandu(Komande komanda, Tabla &polja, int &x, int &y,
  Smjerovi p_smjer = Smjerovi::Gore, int p_x = 0, int p_y = 0);
```

koja poziva neku od funkcija "`Idi`", "`BlokirajPolje`", itd. zavisno od komande koja joj se proslijedi kao prvi parametar. Drugi, treći i četvrti parametar su tabla, odnosno tekuća pozicija igrača respektivno. Eventualni parametri koji su potrebni za izvršenje komande dati su kroz preostale parametre. Izvršavanje bilo koje komande koja dovodi do promjene pozicije igrača treba da proizvede ispis oblika "Tekuca pozicija igrača je (x, y)". Pri tome, ukoliko funkcija "`Idi`" vrati rezultat "`KrajPobjeda`" ili "`KrajPoraz`", treba još ispisati tekst "Bravo, obisli ste sva sigurna polja" odnosno "Nagazili ste na minu" i nakon toga uraditi isto što treba uraditi i ukoliko je proslijeđena komanda "`ZavrsiIgru`". Komanda "`PrikaziOkolinu`" proizvodi ispis matrice sa informacijama o minama u okolici (tj. ono što vrati istoimena funkcija), red po red, sa jednim razmakom između elemenata u jednom redu. Komande za blokadu/deblokadu ne proizvode nikakav ispis. Ukoliko bilo koja od funkcija koja je pozvana iz ove funkcije baci izuzetak, ova funkcija treba prikazati prateći tekst tog izuzetka na ekran. Svaki eventualni ispis iz ove funkcije treba biti praćen prelaskom u novi red (tj. kurzor ne treba ostati u istom redu u kojem je ispisani tekst). Ukoliko je proslijeđena komanda "`ZavrsiIgru`", funkcija treba očistiti matricu od svih modifikacija (postaviti sva polja na "`Prazno`"), te baciti izuzetak tipa "`runtime_error`" uz prateći tekst "Igra zavrsena".

Konačno, ukoliko je proslijeđena komanda "`KreirajIgru`", potrebno je od korisnika tražiti unos broja polja te nakon toga pozicije mina. Pozicije mina se unose u formatu "(x, y)", a unos treba završiti tačkom. U slučaju unosa u pogrešnom formatu treba ignorirati potrešno uneseni podatak, ispisati tekst "Greska, unesite ponovo!" i od korisnika ponovo tražiti unos. Ista stvar treba da se desi ukoliko neka od koordinata nije smislena niti legalna. Unesene podatke treba proslijediti funkciji "`KreirajIgru`". Slijedi primjer dijaloga koji proizvodi ova komanda:

```
Unesite broj polja: 5
Unesite pozicije mina: (1,2)
(3,3)
(3,7)
Greska, unesite ponovo!
(3,4)
d
Greska, unesite ponovo!
(4,0)
.
```

Napisane funkcije povežite u jednu cjelinu u glavnom programu koji će u petlji ispisivati tekst "Unesite komandu: " te pozvati funkciju "`UnosKomande`". Ukoliko je unos korektan, treba izvršiti komandu pozivom funkcije "`IzvrsiKomandu`", u suprotnom treba prijaviti grešku pozivom funkcije "`PrijaviGresku`". Sve ovo se ponavlja sve dok se igra ne završi (tj. dok ne dođe do bacanja izuzetka koji završava igru). Kada se to desi, program treba da ispiše tekst "Dovidjenja!" i da završi sa radom.

2. Napišite funkciju "`IzdvojiDijagonale3D`" sa dva parametra, čiji je prvi parametar neki 3D kontejner "`kont`", a drugi parametar je pobrojanog tipa "`SmjerKretanja`". Funkcija kao rezultat vraća 2D kontejner istog tipa kao što su elementi 3D kontejnera "`kont`" poslanog kao prvi parametar, a koji se sastoji od svih elemenata glavnih dijagonala koje se nalaze u pojedinačnim elementima 3D kontejnera "`kont`", poredane po redovima (ovo će biti detaljno objašnjeno malo kasnije). Informaciju u kojem smjeru se potrebno kretati prilikom izdvajanja elemenata sa glavnih dijagonala nosi drugi parametar koji je pobrojanog tipa. Kako 3D kontejner možemo vizualizirati kvadrom u prostoru, jasno je da se možemo kretati u 6 smjerova: sprijeda unazad; odozada unaprijed; odozgo prema dolje; odozdo prema gore; slijeva nadesno i zdesna na lijevo. Ovi smjerovi su respektivno definisani u sljedećem pobrojanom tipu:

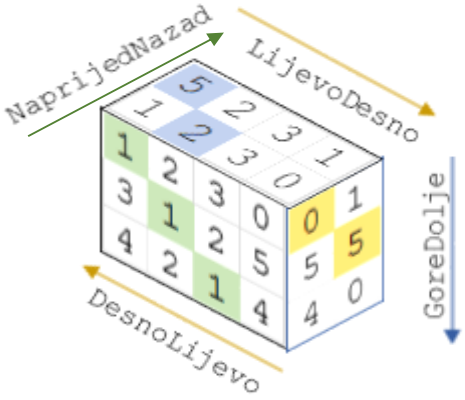
```
enum class SmjerKretanja {
   NaprijedNazad, NazadNaprijed, GoreDolje, DoljeGore, LijevoDesno, DesnoLijevo
};
```

Kontejner "kont" poslan kao prvi parameter funkcije je potpuno generički, odnosno nije unaprijed specificirano od kojih tipova je izgrađen (može biti *vektor dekôvā vektôrā cijelih brojeva*, *dek dekôvā dekôvā znakova* itd.), ali je poznato da podržava funkcije "size", "at" i "push_back", pri čemu sve ostalo ne mora biti podržano (pa ni indeksiranje). Isto vrijedi i za sve "potkontejnere" (2D i 1D kontejnere) od kojih je ovaj 3D kontejner građen. Za elemente kontejnera (npr. "**int**", "**double**", "MojTip" itd) pretpostavlja se da mora biti podržano samo kopiranje i dodjeljivanje, dok druge operacije (kao što su recimo sabiranje itd.) ne moraju nužno biti podržane.

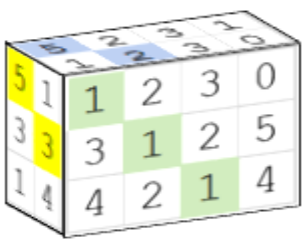
Funkcija vraća 2D "potkontejner" istog tipa kao i jedan elemenat 3D kontejnera "kont" (npr. ako je "kont" tipa *vektor dekôvā vektôrā cijelih brojeva* funkcija treba da vrati *dek vektôrā cijelih brojeva*) koji sadrži sve dijagonale, poredane po redovima (prvi elemenat dijagonale je prvi elemenat reda) svih 2D potkontejnera. Na primjer, uzmimo da je "kont" dek vektôrā vektôrā cijelih brojeva formata $2 \times 3 \times 4$ prikazan ispod:

$$kont = \left[\begin{pmatrix} 1 & 2 & 3 & 0 \\ 3 & 1 & 2 & 5 \\ 4 & 2 & 1 & 4 \end{pmatrix} \begin{pmatrix} 5 & 2 & 3 & 1 \\ 3 & 5 & 2 & 5 \\ 1 & 7 & 7 & 0 \end{pmatrix} \right]$$

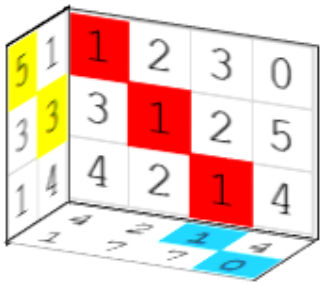
Za ovakav kontejner funkcija može, ovisno od toga kakav je drugi parametar, vratiti 6 različitih dvodimenzionalnih potkontejnera sastavljenih od dijagonala vektora vektôrā koji su elementi 3D kontejnera. Radi lakšeg objašnjenja šta se tačno treba uraditi, korisno je ovaj kontejner vizualizirati kao kvadar u 3 dimenzije (pri čemu se neće vidjeti drugi potkontejner u potpunosti):



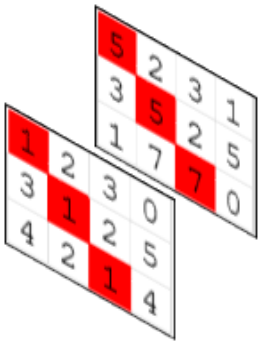
Smjer "NaprijedNazad" (na slici je glavna dijagonala označena zelenom bojom) sadrži elemente sa glavne dijagonale ukoliko se prvo gleda prvi 2D potkontejner pa onda drugi itd. Slično, smjer "NazadNaprijed" će kao svoj prvi elemenat sadržavati elemente glavne dijagonale posljednjeg 2D kontejnera gledajući sa kraja (to je sporedna dijagonala sa „druge strane" matrice na slici iznad sa elementima 1, 2 i 7, samo se prvi elemenat vidi na slici), a kao drugi elemenat (red) će sadžavati elemente glavne dijagonale pretposljednjeg (u ovom slučaju prvog) 2D kontejnera (elementi 0, 2 i 2 što se može vidjeti na prvom potkontejneru sa slike). Analogno se definišu i ostali smjerovi, npr. za smjer "DesnoLijevo" (na slici označen žutom bojom) prva dijagonala se sastoji od elemenata 0 i 5 i to treba da bude prvi elemenat (red) povratnog 2D kontejnera, a za obrnuti smjer, "LijevoDesno", red sa elementima 5 i 3 treba da bude posljednji red povratnog 2D kontejnera. Da bi se stvorila jasnija slika dijagonala iz svih pravaca kretanja, prikazaćemo još nekoliko slika. Prikaz sa lijeve strane (gdje se vidi prvi red 2D kontejnera kojeg funkcija vraća ako se kao drugi parametar proslijedi "LijevoDesno") izgleda ovako:



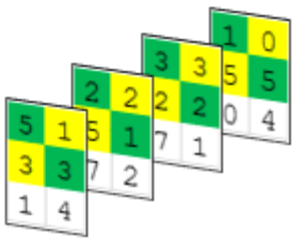
Ukoliko izvršimo prikaz sa donje strane možemo vidjeti markirane elemente 0 i 1 koji čine prvi red povratnog kontejnera kada se kao drugi parametar proslijedi "`DoljeGore`". Primijetimo da kod "suprotnih" smjerova (npr. smjerova "`LijevoDesno`" i "`DesnoLijevo`" odnosno "`DoljeGore`" i "`GoreDolje`") imamo i suprotne dijagonale: ako je iz perspektive "`GoreDolje`" prva glavna dijagonala sa elementima 5 i 2, a prva sporedna ima elemente 1 i 3, (može se vidjeti na slici iznad) to je iz suprotne perspektive "`DoljeGore`" upravo dijagonala sa elementima 1 i 3 posljednja *glavna* dijagonala. Prikažimo sada i sa donje strane kontejner:



Radi kompletnosti, prikažimo 3D kontejner i kao dva uzastopna 2D kontejnera:



Crvenom bojom su označene dijagonale koje treba smjestiti u rezultat ukoliko se kao smjer kretanja pošalje "`NaprijedNazad`", pri čemu rezultantni kontejner kao prvi red treba da sadrži elemente 1, 1 i 2, a kao drugi red 5, 5 i 7. Konačno, prikažimo i 3D kontejner kao 4 uzastopna 2D kontejnera iz smjera "`LijevoDesno`". Glavne dijagonale su prikazane zelenom bojom, ali su na istoj slici žutom bojom prikazane sporedne dijagonale, koje su zapravo glavne dijagonale za suprotan smjer "`DesnoLijevo`":



Kao rezime, prikažimo sve moguće 2D potkontejnere kontejnera "`kont`" iz navedenog primjera koje bi funkcija mogla vratiti, ukoliko joj proslijedimo redom sve moguće vrijednosti koje drugi parametar može imati:

$$NaprijedNazad = \begin{bmatrix} 1 & 1 & 1 \\ 5 & 5 & 7 \end{bmatrix};\quad NazadNaprijed = \begin{bmatrix} 1 & 2 & 7 \\ 0 & 2 & 2 \end{bmatrix};\quad GoreDolje = \begin{bmatrix} 5 & 2 \\ 3 & 1 \\ 1 & 2 \end{bmatrix}$$

$$DoljeGore = \begin{bmatrix} 0 & 1 \\ 5 & 2 \\ 1 & 3 \end{bmatrix};\quad LijevoDesno = \begin{bmatrix} 5 & 3 \\ 2 & 1 \\ 3 & 2 \\ 1 & 5 \end{bmatrix};\quad DesnoLijevo = \begin{bmatrix} 0 & 5 \\ 3 & 2 \\ 2 & 5 \\ 1 & 3 \end{bmatrix}$$

Pretpostavka se da kontejner ima oblik savršenog kvadra, odnosno da svaka njegova ploha (2D potkontejner) ima oblik klasične matrice (isti broj elemenata u svakom redu), te da su sve plohe iste dimenzije. Ukoliko nije ispunjeno da svi redovi u svim plohama imaju isti broj elementa, funkcija treba baciti izuzetak tipa "`domain_error`" uz prateći tekst "Redovi nemaju isti broj elemenata". Ukoliko je ovaj uvjet ispunjen, ali nije ispunjen uvjet da sve plohe imaju isti broj redova, funkcija treba baciti izuzetak istog tipa, ali uz prateći tekst "Plohe nemaju isti broj redova".

Napomena: Ovaj zadatak je više logička glavolomka nego programerski problem. Ukoliko se "pametno" pristupi zadatku, odnosno ukoliko se na elegantan način postave granice petlji te pristupi elementima kontejnera, zadatak se može uraditi u manje od 40 linija kôda.

Napisanu funkciju testirajte u glavnom program gdje se sa tastature unose tri cijela broja koja predstavljaju dimenzije vektora dekôvā dekôvā cijelih brojeva, zatim elementi tog kontejnera (plohu po plohu, a red po red unutar jedne plohe), te cijeli broj u opsegu [0, 5] koji predstavlja smjer kretanja (0 = "`NaprijedNazad`", 1 = "`NazadNaprijed`", itd.). Po okončanju unosa, program treba pozvati funkciju "`IzdvojiDijagonale3D`" i ispisati skup traženih dijagonala. Obavezno predvidite hvatanje svih izuzetaka koje funkcija može baciti (iako se to u Vašoj glavnoj funkciji nikad neće desiti jer se uvijek unosi ispravan kontejner). Širinu ispisa podesite na 4 mjesta. Slijedi primjer dijaloga između programa i korisnika (eventualni drugi dijalozi biće specificirani javnim autotestovima):

```
Unesite dimenzije (x y z): 2 3 4
Unesite elemente kontejnera:
1 2 3 0
3 1 2 5
4 2 1 4

5 2 3 1
3 5 2 5
1 7 7 0
Unesite smjer kretanja [0-5]: 3
DoljeGore:
   0   1
   5   2
   1   3
```

Napomena: novi red između dvije matrice u unosu je proizvoljan, ulazni tok će ga svakako ignorirati, pa nije potrebna posebna logika prilikom unosa kontejnera.

3. Napišite generičke funkcije nazvane "`UvrnutiPresjek`" i "`UvrnutaRazlika`".

Funkcija "`UvrnutiPresjek`" prihvata pet parametara, od kojih su prva dva parametra pokazivači ili iteratori koji pokazuju na početak odnosno tačno iza kraja nekog bloka podataka (tipično unutar nekog kontejnerskog tipa podataka), a treći i četvrti parametar pokazivači odnosno iteratori koji pokaziju na početak odnosno tačno iza kraja nekog drugog bloka podataka. Tip prva dva parametra je isti, a isto vrijedi i za tip trećeg i četvrtog parametra, koji može biti različit od tipa prva dva parametra. Peti parametar je funkcija kriterija, nazovimo je f, koja prihvata jedan parametar proizvoljnog tipa, a povratna vrijednost joj je također proizvoljnog tipa. Za blokove se podrazumijeva da predstavljaju elemente nekih skupova, tako da nema ponavljanja elemenata. Pod "uvrnutim presjekom" skupova S_1 i S_2 smatra se skup uređenih parova (x, y) takvih da je x iz skupa S_1 a y iz S_2, pri čemu se vrijednosti funkcije f poklapaju za x i y (tj. takvih da je $f(x) = f(y)$). Recimo, ukoliko je $S_1 = \{21, 22, 23\}$ i $S_2 = \{12, 44, 32\}$, a f funkcija koja računa sumu cifara svog parametra, tada je uvrnuti presjek skup $\{(21, 12), (23, 32)\}$. Funkcija "`UvrnutiPresjek`" treba kreirati matricu formata $n \times 3$ pri čemu je n broj elemenata u presjeku, dok svaki red matrice sadrži trojku (x, y, f) gdje je (x, y) element uvrnutog presjeka, a f je zajednička vrijednost funkcije koju x i y dijele. Recimo, za prethodni primjer, redovi tražene matrice trebali bi glasiti $\{21, 12, 3\}$ i $\{23, 32, 5\}$. Prije vraćanja tako kreirane matrice kao rezultata iz funkcije, potrebno je njene redove sortirati u rastući poredak po vrijednosti treće koordinate, tj. po vrijednosti funkcije f. Ukoliko dva reda imaju istu vrijednost f, tada ih treba sortirati po vrijednosti x, a ukoliko se i vrijednoti x poklapaju, onda po vrijednosti y. Za sortiranje obavezno koristite funkciju "`sort`" iz biblioteke "`algorithm`" uz prikladnu funkciju kriterija implementiranu kao lambda funkcija.

Funkciju "`UvrnutiPresjek`" treba biti moguće pozvati i sa samo četiri parametra, pri čemu se tada računa obični presjek skupova. U izlaznoj matrici tada treba popuniti samo prvu kolonu elementima presjeka, dok ostale kolone treba popuniti nulama.

Funkcija "`UvrnutaRazlika`" prihvata iste parametre kao i funkcija "`UvrnutiPresjek`", ali kao rezultat vraća matricu formata $n \times 2$. Uvrnutu razliku čine oni elementi iz jednog ili drugog bloka koji nemaju svog para u odnosu na funkciju f, odnosno oni elementi x iz jednog ili drugog bloka za koji ne postoji nikakav element y u suprotnom bloku za koji je $f(x) = f(y)$. Recimo, neka su ponovo dati skupovi $S_1 = \{21, 22, 23\}$ i $S_2 = \{12, 44, 32\}$ i neka je f ponovo funkcija koja računa sumu cifara svog argumenta. Broj 22 čija je suma cifara 4 iz skupa S_1 nema svog para istih svojstava u skupu S_2. Slično vrijedi za broj 44 iz skupa S_2 (on nema svog para u skupu S_1). Svi ostali brojevi iz S_1 Stoga uvrnutu razliku ova dva skupa čine elementi 22 i 44. Te elemente treba smjestiti u prvu kolonu rezultujuće matrice, a u drugu kolonu odgovarajuće vrijednosti funkcije f. Drugim riječima, rezultantna matrica treba biti $\{\{22, 4\}, \{44, 8\}\}$. Međutim, prije njenog vraćanja iz funkcije, njene redove je potrebno sortirati u opadajući poredak po prvom elementu redova, a ako su prvi elementi jednaki, onda po drugom elementu.

Za potrebe testiranja, potrebno je implementirati još nekoliko jednostavnih funkcija koje će se slati kao parametri u funkcije "`UvrnutiPresjek`" i "`UvrnutaRazlika`" (što ne znači da se ovim funkcijama u autotestovima neće slati i druge funkcije), Sve ove funkcije primaju parametar tipa "`long long int`", a vraćaju rezultat tipa "`int`":

"`SumaDjelilaca`": Funkcija vraća sumu svih djelilaca broja koji joj je prenesen kao parametar.
"`BrojProstihFaktora`": Funkcija vraća broj svih prostih faktora broja koji joj je prenesen kao parametar. Recimo, za ulaz 21 funkcija treba da vrati 2, jer 27 ima 2 prosta faktora (3 i 7).
"`BrojSavrsenihDjelilaca`": Funkcija vraća broj savršenih djelilaca broja, tj. djelilaca koji su savršeni brojevi (npr. za broj 168 funkcija treba da kao rezultat vrati 2, jer broj 168 ima dva prosta djelioca: 6 i 28). Podsjetimo se da su savršeni brojevi oni koji su jednaki sumi svih svojih djelilaca, ne računajući njih same.

Napisane funkcije testirajte u glavnom programu koji će prvo tražiti da se sa tastature unesu elementi dva deka (pri čemu se broj elemenata prethodno unosi sa tastature), Pri unosu treba obezbijediti da se ponavljanje već unesenog elementa ignorira, tako da oba deka sigurno imaju sve različite elemente prije poziva funkcija. Nakon toga treba pozvati odgovarajuće funkcije, te ispisati uvrnuti presjek računat prema sumi cifara, te uvrnutu razliku prema broju prostih faktora. Dijalog između korisnika i programa treba izgledati ovako (rezervirajte 6 mjesta za ispis elemenata matrice):

```
Unesite broj elemenata prvog kontejnera: 6
Unesite elemente prvog kontejnera: 15 16 17 12 13 14 15
Unesite broj elemenata drugog kontejnera: 8
Unesite elemente prvog kontejnera: 31 51 91 71 21 31 51 23 24 22
Uvrnuti presjek kontejnera:
    12      21      3
    13      31      4
    13      22      4
    14      23      5
    15      51      6
    15      24      6
    17      71      8
Uvrnuta razlika kontejnera:
    16       7
    91      10
Dovidjenja!
```

4. Predmetni nastavnik ima neka saznanja vezano za kupovine zadaća iz Tehnika programiranja. Svakodnevno putem sofisticiranih kanala izvještavanja on dobija nove informacije o slučajevima prepisivanja i kupovine zadaća (pristigle informacije će biti iskorištene za kreiranje skrivenih autotestova za ovaj zadatak). S obzirom na ogromnu količinu informacija, potrebna mu je pomoć pri evidentiranju istih i pronalasku krivaca. Napravite program koji će omogućiti jednostavniju analizu pomoću funkcija koje su date u nastavku.

Funkcija "`PotencijalniKrivci`" prima dva parametra: prvi je referenca na dvostruki pokazivač na znakove, a drugi vektor stringova. U vektoru stringova se funkciji prosljeđuje spisak imena potencijalnih "prodavača" zadaća, pri čemu se smatra da je svaki string legalno ime (dakle, neko se može zvati i " &&%* Tarzan 007/+12" . Funkcija treba dinamički alocirati niz pokazivača na dinamički alocirane nizove znakova u koje će se smjestiti imena proslijeđena funkciji, te u prvi parametar funkcije smjestiti pokazivač na početak ovog niza. Za svaki od nizova znakova treba alocirati tačno onoliko prostora koliko treba, uključujući i prostor za NUL graničnik (oznaka kraja C-ovskog stringa). Kao rezultat funkcija treba da vrati broj alociranih nizova znakova. U slučaju da alokacija ne uspije, treba baciti izuzetak tipa "`bad_alloc`", ali pri tome ne smije da dođe do curenja memorije ni u kom slučaju.

Funkcija "`OdbaciOptuzbu`" prima tri parametra. Prva dva parametra su referenca na dvostruki pokazivač koji ima isto značenje kao i prvi parametar u prethodnoj funkciji, drugi parametar je veličina niza pokazivača koji pokazuju na potencijalne "prodavače", dok je treći parametar tipa string. Ovaj parametar predstavlja ime osobe za koju se pouzdano utvrdilo da nije prodavala zadaću, te je tu osobu potrebno izbrisati iz evidencije potencijalnih krivaca. Ukoliko se ustanovi da takva osoba uopće nije u popisu potencijalnih krivaca, treba baciti izuzetak tipa "`domain_error`" uz prateći tekst "Osoba sa imenom <*ime*> nije bila optuzena", pri čemu će umjesto <*ime*> pisati traženo ime koje je poslano funkciji. U suprotnom, treba osobu izbrisati iz spiska, odnosno "osloboditi je optužbe". To treba izvesti ovako. Prvo treba obrisati niz znakova koji sadrži podatke o osobi, a nakon toga odgovarajući pokazivač u nizu pokazivača postaviti na nul-pokazivač, kao indikator da je osoba obrisana. Ukoliko nakon toga ne bude više od 10 nul-pokazivača u nizu pokazivača (tj. ukoliko do tada nije izbrisano više od 10 osoba), postupak je gotov. U suprotnom, treba obaviti "uštedu" memorije, uklanjanjem prostora koji zauzimaju nul-pokazivači, tako što će se alocirati novi niz pokazivača, onolike veličine koliko zaista ima osoba u spisku (ne brojeći obrisane osobe), nakon čega će se svi "korisni" pokazivači prekopirati u novi niz, stari niz pokazivača će se ukloniti, a pokazivač na novi niz će se dodijeliti prvom parametru. U slučaju da ne uspije alokacija novog niza pokazivača, ne treba da se desi ništa (ne treba ni da se baci izuzetak), sve treba ostati kako je bilo. Funkcija kao rezultat vraća novu veličinu niza pokazivača (to može da bude i ista veličina kakva je bila, ukoliko nije došlo do alokacije novog niza pokazivača).

Funkcija "`DodajOptuzbu`" prima iste parametre kao i funkcija "`OdbaciOptuzbu`", s tim što ona vrši dodavanje novog imena u niz potencijalnih krivaca. Postupak teče ovako. Treba alocirati novi niz znakova koji je dovoljno dug da prihvati ime, plus jedno mjesto za NUL graničnik, i iskopirati ime u taj prostor. Zatim treba pretražiti niz pokazivača. Ukoliko u njemu ima neki nul-pokazivač, na prvo mjesto u nizu gdje se nalazi nul-pokazivač treba upisati adresu novokreiranog niza znakova i postupak je gotov. U suprotnom (tj. ukoliko su svi pokazivači "zauzeti"), treba alocirati novi niz pokazivača koji je za jedno mjeto veći nego trenutni niz pokazivača, zatim prepisati sve pokazivače iz starog niza u novi, na posljednje mjesto u novi niz upisati adresu kreiranog niza znakova, obrisati stari niz pokazivača, i na kraju dodijeliti adresu novog niza pokazivača prvom parametru. U slučaju da neka od traženih alokacija ne uspije, funkcija treba baciti izuzetak tipa "`bad_alloc`", ali stanje memorije nakon bacanja izuzetka treba da bude posve isto kao da funkcija nije uopće ni bila pozvana. Funkcija kao rezultat vraća novu veličinu niza pokazivača.

Posljednja funkcija koju trebate implementirati je "`IzlistajOptuzbu`" koja prima dva parametra, koji su isti kao i prva dva parametra u prethodnim funkcijama, samo što prvi parametar ne treba da bude referenca. Ova funkcija vrši ispis svih registriranih imena, svako u novom redu (ukoliko je neki od pokazivača u nizu pokazivača nul-pokazivač, treba ga ignorirati, jer on odgovara osobi koja je obrisana). Funkcija ne vraća nikakav rezultat.

Napisane funkcije demonstrirajte u kratkom testnom programu. Eventualni dijalog između korisnika i programa biće specificiran javnim autotestovima.