Zadaća 3.

Ova zadaća nosi ukupno 5 poena. Prvi i drugi zadatak nose po 1 poen, treći i četvrti zadatak nose po 1,2 poena, dok posljednji zadatak nosi 0,6 poena. Svi zadaci se mogu uraditi na osnovu gradiva sa prvih osam predavanja i pretpostavljenog predznanja iz predmeta "Osnove računarstva". Rok za predaju ove zadaće je utorak, 3. V 2016. do 20.00.

1. Napišite generičke funkcije "`ProslijediLijevo`", "`ProslijediDesno`" i "`MonstrumFunkcija`" koje rade sljedeće stvari. Funkcija "`ProslijediLijevo`" je takva da poziv

   `ProslijediLijevo(`f`)(`x`, `y`, `z`)`

   prosljedi parametre x, y, z funkciji f. Pri tome, ovaj poziv treba da kao rezultat da istu vrijednost kakvu bi dala funkcija f kada joj se pošalju argumenti x, y, z, tj. vrijednost izraza $f(x, y, z)$. Slično, funkcija "`ProslijediDesno`" također prosljeđuje parametre x, y i z funkciji f, samo što se parametri ovaj put nalaze sa lijeve strane u odnosu na funkciju f i prosljeđuju se desno. Drugim riječima, poziv koji bi proizveo isti rezultat kao i prethodni izgledao bi ovako:

   `ProslijediDesno(`x`, `y`, `z`)(`f`)`

   Za obje ove funkcije, parametar f je funkcija koja prihvata tri parametra istog tipa i vraća neku vrijednost istog tipa kao i parametri. Za realizaciju koristite polimorfne funkcijske omotače (tj. tip "`std::function`"). Slijede primjeri nekih poziva ove dvije funkcije:

   ```
   std::string s1("Ja"), s2("Volim"), s3("TP");
   std::cout << ProslijediLijevo<std::string>(Spoji)(s1, s2, s3) << std::endl;
   std::cout << ProslijediDesno(s1, s2, s3)(Spoji);
   ```

   Ovaj isječak treba da dva puta ispiše "JaVolimTP", svaki ispis u posebnom redu. Funkcije trebaju da kao argumente mogu prihvatiti i obične (imenovane) funkcije i anonimne (lambda) funkcije. Razlog za eksplicitno navođenje tipa prilikom poziva funkcije "`ProslijediLijevo`", tj, pisanje konstrukcije poput "`ProslijediLijevo<std::string>`" leži u jednom ograničenju vezanom za dedukciju kod generičkih funkcija. Naime, kako je rečeno da se za realizaciju koriste polimorfni funkcijski omotači, jasno je (odnosno, bar bi trebalo da bude jasno) da će parametar funkcije "`ProslijediLijevo`" biti polimorfni funkcijski omotač, i to kod kojeg tip parametara i povratne vrijednosti nisu unaprijed poznati, nego se određuju dedukcijom (očito je u pitanju djelimična dedukcija). Nažalost, kod dedukcije u generičkim funkcijama, kao što je poznato, nema nikakvih improvizacija (poput automatskih pretvorbi tipova), nego se mora uspostaviti *potpuno slaganje tipa* da bi dedukcija bila uspješna. To znači da bi dedukcija uspjela, ono što šaljemo funkciji kao parametar *mora zaista biti polimorfni funkcijski omotač* (tj, nešto deklarirano sa "`std::function`"), a ne nešto što se može automatski pretvoriti u njega. U našem primjeru "`Spoji`" *nije polimorfni funkcijski omotač nego funkcija* i dedukcija ne uspijeva (bez obzira što se funkcija može automatski pretvoriti u polimorfni funkcijski omotač i dodijeliti njemu). Da bi se izbjegao ovaj problem, treba koristiti *eksplicitnu specifikaciju tipa* i tako zaobići dedukciju. Dedukcija bi radila (i navođenje tipa ne bi bilo potrebno) da je "`Spoji`" bio *zaista deklariran kao polimorfni funkcijski omotač* kome je dodijeljena neka funkcija (imenovana ili lambda), odnosno da smo imali nešto poput

   ```
   std::function<std::string(std::string, std::string, std::string)> Spoji =
     [](std::string s1, std::string s2, std::string s3) { return s1 + s2 + s3; }
   ```

   Ukoliko ste shvatili šta je gore rečeno, onda bi Vam trebalo biti jasno zbog čega ove "vratolomije" nisu potrebne kod funkcije "`ProslijediDesno`" (pomoć: razmislite šta su parametri ove funkcije). Ipak, i ovdje će ponekad biti potrebno zaobići dedukciju, kao u sljedećem primjeru koji bi na ekranu trebao ispisati "Tehnike programiranja":

   ```
   std::cout << ProslijediDesno<std::string>("Tehnike", " ",
     "programiranja")(Spoji);
   ```

   Eksplicitno navođenje tipa je ovdje potrebno s obzirom da su stringovni literali poput "Tehnike" zapravo tipa "`const char []`" a ne "`std::string`". Dedukcija će upravo to i zaključiti, ali kasnije će nastati problem kada joj se bude slala funkcija "`Spoji`" čiji su parametri tipa "`std::string`", što je suprotno očekivanjima da su parametri tipa "`const char []`". Stoga je rješenje da se i ovaj put zaobiđe dedukcija.

Ove dvije funkcije su bile samo lagani uvod (zagrijavanje) za treću funkciju koja nosi glavninu poena na ovom zadatku, a koja se zove "`MonstrumFunkcija`". Ona treba biti takva da poziv

```
MonstrumFunkcija(f)(x)(y, z)(g)(r)
```

treba prvo da funkciji f (koja prihvata tri parametra) proslijedi parametre x, y i z, a zatim rezultat tog poziva $f(x, y, z)$ i parametar r treba proslijediti funkciji g (koja prihvata dva parametra). U suštini, u ovakvom pozivu, funkcija g kao prvi parametar prihvata sve što je lijevo od nje (tj. $f(x, y, z)$), a kao drugi parametar ono što je desno od nje (r), odnosno prosljeđivanje funkciji g se vrši i s lijeve i s desne strane (ipak, nemojte pokušavti za ovu svrhu koristiti prethodno napisane dvije funkcije, neće Vam biti od koristi). Rezutat ovog poziva je ono što na kraju vrati funkcija g. Pri tome, svi parametri x, y, z i r su *istog tipa*.

Funkciju g ćemo nazvati *korektivna funkcija* jer ona prihvata rezultat funkcije f, transformira ga, te tako modificirani rezultat vraća kao svoju povratnu vrijednost (koja je ujedno i povratna vrijednost čitavog prikazanog poziva funkcije "`MonstrumFunkcija`"). Na primjer, neka funkcija f računa zbir svoja 3 argumenata x, y, i z, a neka funkcija g korigira taj zbir (računanjem ostatka pri dijeljenju) tako da je on uvijek u opsegu od 0 do r, pri čemu je r njen drugi parametar. Navedeno se može postići sa sljedećim kôdom, pri čemu funkcija g korigira rezultat funkcije f da bude u intervalu [0, 100):

```
std::function<int(int, int, int)> Saberi =
  [](int x, int y, int z) { return x + y + z; };
std::function<int(int, int)> Korigiraj = [] (int x, int y) { return x % y;};
int x, y, z; std::cin >> x >> y >> z;
std::cout << MonstrumFunkcija (Saberi)(x)(y, z)(Korigiraj)(100);
```

Za unesene vrijednosti 62, 41 i 4 isječak ispisuje broj 7 jer je zbir $62 + 41 + 4 = 107$, a to nakon korigiranja uzimanjem ostatka pri dijeljenju sa 100 postaje broj 7 jer je $107 \% 100 = 7$. Slijedi još jedan primjer:

```
auto PomnoziDodaj7 = [](int x, int y) { return x * y + 7;};
auto Saberi = [](int x, int y, int z) { return x + y + z;};
std::cout << MonstrumFunkcija<int>(Saberi)(12)(5, 60)(PomnoziDodaj7)(10);
```

Isječak iznad ispisuje "777" jer je $12 + 5 + 60 = 77$, i to je rezultat funkcije "`Saberi`" koji se prosljeđuje funkciji "`PomnoziDodaj7`" funkciji koja ga pomnoži sa svojim parametrom 10 i doda 7 na rezultat množenja. Ovdje također imamo eksplicitnu specifikaciju tipa "`int`" prilikom poziva funkcije "`MonstrumFunkcija`" (tj. navodimo "`MonstrumFunkcija<int>`"). Razlog leži u činjenici da su funkcije "`Saberi`" i "`PomnoziDodaj7`" definirane pomoću ključne riječi "`auto`" kod koje, slično kao kod dedukcije u generičkim funkcijama, također nema nikakve improvizacije (tačnije, ključna riječ "`auto`" zapravo i vrši dedukciju), tako da će tip promjenljive biti određen tačno onakav kakav je tip onoga čime je inicijaliziramo. Kako je u ovom primjeru inicijaliziramo lambda funkcijom, "`Saberi`" i "`PomnoziDodaj7`" će po tipu biti lambda funkcije, a ne polimorfni funkcijski omotači, i dedukcija neće uspjeti, iz istih razloga zbog kojih nije uspjevala i kada smo navodili primjer za funkciju "`ProslijediLijevo`" (dakle, ne uspijeva zbog toga što lambda funkcija *nije polimorfni funkcijski omotač*, mada se može *automatski konvertirati u njega*).

Potrebno je također omogućiti izostavljanje funkcije g na način da se kao odgovarajući parametar proslijedi nul-pokazivač. U tom slučaju opisani poziv funkcije "`MonstrumFunkcija`" treba da vrati isti rezultat kao i funkcija f. Ovo se koristi kada ne želimo korigirati povratnu vrijednost funkcije f. Kao parametar r se u tom slučaju navodi bilo koja vrijednost jer se ona ignorira. Recimo, treba omogućiti poziv poput sljedećeg:

```
std::cout << MonstrumFunkcija<int>(Saberi)(10)(1, 2)(nullptr)(0);
```

Isječak iznad ispisuje 13. Nažalost, nije jednostavno izbjeći navođenje nul-pokazivača i parametra r (0 u primjeru iznad) jer u C++11 standardu lambda funkcije ne mogu imati parametre sa podrazumijevanim vrijednostima (ali je ta mogućnost uvedena u standard C++14).

Demonstrirajte rad napisanih funkcija u "`main`" funkciji tako što ćete sa tastature unijeti tri realna broja te proslijediti ih lijevo (koristeći funkciju "`ProslijediLijevo`") funkciji "`ZbirSinusa`" (implementiranu kao imenovana funkcija) koja računa vrijednost izraza $\sin x + \sin y + \sin z$ (pri čemu su x, y i z njeni parametri), i proslijediti ih desno (koristeći funkciju "`ProslijediDesno`")

lambda funkciji koja računa vrijednost izraza $\cos x \cdot \cos y \cdot \cos z$. Također, rezultat funkcije "`ZbirSinusa`" treba se pomoću funkcije "`MonstrumFunkcija`" proslijediti korektivnoj funkciji koja množi (skalira) proslijeđeni rezultat sa njenim drugim parametrom (faktorom skaliranja). Dijalog sa korisnikom trebao bi izgledati ovako:

```
Unesite x y z: 1.2 0.5 2.16
Unesite faktor skaliranja: 10
Prosljedjivaje ulijevo: 2.24285
Prosljedjivaje udesno: -0.176712
Monstrum: 22.4285
```

2. Dopunite program za rad sa generičkom strukturom "`Matrica`" obrađen na Predavanju 8_b sa novom funkcijom "`ProsiriPoFunkcijama`". Ova funkcija kao prvi parametar prima matricu čiji su elementi cjelobrojni (tačnije objekat tipa "`Matrica<int>`" gdje je "`Matrica`" razvijena generička struktura), kao drugi parametar prima mapu čiji su ključevi tipa "`string`" a pridružene vrijednosti funkcije (tačnije objekti tipa "`function`") koje primaju jedan cjelobrojni argument (tipa "`int`") i vraćaju cjelobrojni rezultat, dok kao treći parametar prma cijeli broj n (tipa "`int`"). Mapa smije imati samo 3 vrijednosti ključnog polja: "desno", "dolje" i "dijagonalno" (tako da, u suštini, mapa smije sadržavati najviše 3 para). Ukoliko mapa sadrži bilo kakav par sa nekim drugim ključnim poljem, treba baciti izuzetak tipa "`logic_error`" uz prateći tekst "Neispravna mapa!". Funkcija treba da vrati kao rezultat novu matricu dobijenu na sljedeći način. Ukoliko se u mapi nalazi par sa ključem "desno", matrica treba da se proširi zdesna novim elementima koji se dobijaju primjenom funkcije pridružene ključu "desno" nad elementima izvorne matrice (tako da će matrica postati dvostruko šira). Slično, ukoliko se u mapi nalazi par sa ključem "dolje", matricu treba proširiti s donje strane novim elementima koji se dobijaju primjenom funkcije pridružene ključu "dolje". Ukoliko se zahtijeva i proširivanje desno i proširivanje dolje, odgovarajuću "rupu" koja nastaje dijagonalno dolje desno treba popuniti kopijom izvornih elemenata, osim ukoliko je zadan i par sa ključem "dijagonalno". U tom slučaju elemente u toj "rupi" treba popuniti rezultatima primjene funkcije pridružene ključu "dijagonalno". Isto tako, ukoliko je u mapi prisutan par sa ključem "dijagonalno", a nedostaje makar jedan od parova sa ključevima "desno" ili "dolje", popunjavamo ono što možemo popuniti u skladu sa opisanim postupkom, dok "rupe" koje nastaju popunjavamo kopijom izvornih elemenata. Ukoliko je u mapi prisutan samo par sa ključem "desno" ili samo par sa ključem "dolje", proširivanje matrice vršimo samo na jednu stranu (zdesna odnosno odozdo). Konačno, ukoliko je mapa prazna, ne vršimo nikakvo proširivanje.

   Opisani postupak treba obaviti $n-1$ puta, pri čemu se u svakoj i-toj iteraciji proširuje matrica dobijena u $(i-1)$-oj iteraciji. Pri tome treba naglasiti da je zbog *edukativnih* a ne praktičnih razloga, ovaj postupak potrebno obavljati *tačno ovako kao što je opisano*, odnosno u svakoj iteraciji je potrebno kreirati *novu matricu* i popunjavati je na osnovu matrice dobijene u prethodnoj iteraciji (odnosno, nije dozvoljeno odmah na početku kreirati matricu konačne veličine i popunjavati je postupno na osnovu popunjenog jednog njenog dijela). Na primjer, neka je A matrica formata 2×2

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

   i neka je mapa m deklarirana na sljedeći način:

```
std::map<std::string, std::function<int(int)>> m{
  {"desno", [] (int x) { return x + 1; }},
  {"dolje", [] (int x) { return x + 2; }},
  {"dijagonalno", [] (int x) { return x + 3; }}
};
```

   Nakon poziva funkcije "`ProsiriPoFunkcijama(A, m, 3)`", matrica koja se dobije kao rezultat treba imati sljedeću strukturu (ali je pri tome prethodno trebala biti kreirana i matrica formata 4×4, čiji su elementi prikazani podebljano u dolje prikazanoj matrici):

$$\begin{pmatrix} \mathbf{1} & \mathbf{2} & \mathbf{2} & \mathbf{3} & 2 & 3 & 3 & 4 \\ \mathbf{3} & \mathbf{4} & \mathbf{4} & \mathbf{5} & 4 & 5 & 5 & 6 \\ \mathbf{3} & \mathbf{4} & \mathbf{4} & \mathbf{5} & 4 & 5 & 5 & 6 \\ \mathbf{5} & \mathbf{6} & \mathbf{6} & \mathbf{7} & 6 & 7 & 7 & 8 \\ 3 & 4 & 4 & 5 & 4 & 5 & 5 & 6 \\ 5 & 6 & 6 & 7 & 6 & 7 & 7 & 8 \\ 5 & 6 & 6 & 7 & 6 & 7 & 7 & 8 \\ 7 & 8 & 8 & 9 & 8 & 9 & 9 & 10 \end{pmatrix}$$

Sve pomoćne matrice koje su eventualno kreirane u funkciji moraju biti uklonjene prije izlaska iz funkcije (inače im nećemo moći pristupiti niodakle i imaćemo curenje memorije). U slučaju da je $n = 1$, rezultirajuća matrica treba biti ista kao i izvorna. U slučaju da parametar n ima vrijednost manju ili jednaku nuli, funkcija treba baciti izuzetak tipa "`logic_error`" uz prateći tekst "Besmislen parametar n!".

Pored ove dvije funkcije, treba također proširiti funkciju "`IspisiMatricu`" sa dodatna dva parametra nazvana "`preciznost`" i "`treba_brisati`". Parametar "`preciznost`" je cijeli broj koji određuje preciznost ispisa, odnosno broj tačnih cifara pri ispisu (njega zapravo treba proslijediti funkciji "`cout.precision`" ili manipulatoru "`setprecision`"). Ovaj parametar treba da ima podrazumijevanu vrijednost 4. Drugi parametar "`treba_brisati`" je tipa "**`bool`**". Ukoliko ovaj parametar ima vrijednost "**`true`**", funkcija treba da po obavljenom ispisu oslobodi prostor zauzet matricom koja joj je proslijeđena kao parametar, u suprotnom ne treba da radi ništa po tom pitanju. Ovim se omogućava da možemo zadavati pozive poput

```
IspisiMatricu(ZbirMatrica(a, b), 10, 5, true);
```

tako da se oslobađanje memorije koju je zauzela pomoćna matrica koja predstavlja zbir matrica može obaviti bez korištenja pomoćne promjenljive (kao što smo morali u izvornom programu prikazanom na predavanjima). Pri tome, definirajte još da parametar "`treba_brisati`" ima podrazumijevanu vrijednost "**`false`**", tako da ga ne treba navoditi ukoliko nam brisanje ne treba.

Napisane funkcije testirajte u glavnom programu na primjeru matrica cijelih brojeva čije dimenzije i elemente unosi korisnik putem tastature. Za test funkcije "`ProsiriPoFunkcijama`" sa tastature se također unosi i broj n, a za mapu koristite mapu iz prethodno navedenog primjera (pošto dimenzije rezultatnte matrice veoma brzo rastu sa porastom n, za testiranje koristite male vrijednosti n i matrice malog formata, recimo 2×2). U glavnom programu predvidite hvatanje svih izuzetaka koji bi eventualno mogli nastupiti. Također, dobro pazite da nigdje ne dođe do curenja memorije, ni pod kakvim okolnostima (u suštini, to je i *osnovni cilj zadatka*).

Napomena 1: U ovom programu je *izuzetno lako* napraviti curenje memorije (konkretnije, u funkciji "`ProsiriPoFunkcijama`"). Dobro razmislite šta radite, i ne oslanjate se nasumice ni na kakve "testere curenja memorije" i slična pomagala, jer u suprotnom nećete ovladati tehnikama upravljanja memorijom (to i jeste glavni cilj ovog zadatka). Kad shvatite koliko treba razumijevanja da se u ovom zadatku izbjegne curenje memorije, tek tada ćete *zaista cijeniti destruktore*, koji ovakve probleme rješavaju praktično automatski.

Napomena 2: Nepoštovanje zahtjeva da se u svakoj iteraciji u funkciji "`ProsiriPoFunkcijama`" kreira nova matrica biće kažnjeno davanjem 0 poena na cijeli zadatak. Također je najstrože zabranjeno u ovom zadatku korištenje raznih pomoćnih bibliotečkih kontejnerskih tipova podataka kao što su vektori i drugi, jer bi se njihovom upotrebom izgubila poenta onoga šta se ovim zadatkom htjelo demonstrirati.

3. Na našim prostorima najpoznatije angloameričke igraće karte (a u svijetu su u aktivnoj upotrebi još i italijanske te mađarske karte). Ovaj špil karata se sastoji od 52 karte, po trinaest u svakoj od četiri boje. Boje su herc, karo, pik i tref. Karte su poredane po hijerarhiji: 2, 3, 4, 5, 6, 7, 8, 9, 10, J (Džandar), Q (Dama), K (Kralj), A (As). Vaš zadatak je da korištenjem listi (tj. bibliotečkog kontejnerskog tipa podataka "`list`") čiji su elementi skupovi (tj. tipa "`set`") napravite simulaciju izbacivanja karata iz špila. Lista treba biti sačinjena od ukupno 13 skupova, a pojedinačni elementi u svakom od skupova na početku trebaju biti svi elementi pobrojanog tipa "`Boje`", koji je (na globalnom nivou) definiran kao

```
enum class Boje {Herc, Karo, Pik, Tref};
```

(uz pretpostavku da je špil karata cjelovit). Za tu svrhu, u programu treba implementirati funkcije nazvane "`KreirajSpil`", "`IzbaciKarte`", "`VratiPosljednjihNKarata`" i "`PrebrojiKarte`".

Funkcija "`KreirajSpil`" nema parametara a vraća kao rezultat kompletan kreiran špil (tj. listu od 13 skupova koji sadrže sve karte). Funkcija "`IzbaciKarte`" prihvata dva parametra: referencu na špil (dakle listu skupova), koji *ne mora biti kompletan* (dakle, neke karte mogu nedostajati), te referencu na konstantni uređeni par (tj. bibilotečki tip "`pair`") čije su obje koordinate cjelobrojne (tipa "`int`"), a koji opisuje šta treba izbaciti. Funkcija kao rezultat vraća stek (tj. kontejnerski tip "`stack`") uređenih parova koji će sadržavati izbačene karte (tačan opis šta ovaj stek treba da sadrži naveden je malo kasnije). U parametru koji definira šta treba izbaciti prva koordinata treba biti neki cijeli broj od 1 do 52, a druga koordinata označava broj karata koje je potrebno izbaciti. U slučaju da je neka od koordinata elementa neispravna (tj. ako je prvi parametar manji od 1 ili veći od 52, ili drugi parametar manji od 1), funkcija baca izuzetak tipa "`logic_error`", uz prateći tekst "Neispravni elementi za izbacivanje!".

Način na koji funkcija izbacuje elemente iz liste je sljedeći. Kako se elementi skupova interno čuvaju u sortiranom poretku, redoslijed elemenata u svakom skupu liste je herc, karo, pik i tref (u skladu sa poretkom u tipu "`Boje`"). Prilagodićemo se ovom poretku, tako da ćemo smatrati da prvih 13 elemenata špila čine karte sa bojom herc, sljedećih 13 elementi sa bojom karo itd. U paru koji opisuje šta treba izbaciti prva koordinata je *korak razbrajanja* (r), a druga koordinata *broj karata koji treba izbaciti* (b). Potrebno je kružnim razbrajanjem izbacivati svaku r-tu kartu, sve dok se ne izbaci ukupno b karata, pri čemu se izbačene karte smještaju na stek. Dakle, prvo se izbacuje karta na r-toj poziciji, zatim sljedeća r-ta po redu karta brojano počevši od pozicije gdje je izbačena prethodna karta, i dako dalje. Recimo ukoliko je $r = 12$ i $b = 6$, tada će prva izbačena karta biti dvanaesta po redu u boji herc (to je dakle kralj), sljedeća po redu će biti karo dama, nakon nje pik džandar, nakon nje tref 10, nakon nje herc 9, i konačno na kraju karo 9. Na kraju je, naravno, posljednja izbačena karta na vrhu steka. Stek treba da bude kontejner parova čija je prva koordinata tekstualni naziv boje koja je izbačena (tipa "`string`", pri čemu su legalni nazivi boja "Herc", "Karo", "Pik" i "Tref"), a druga koordinata vrijednost izbačene karte (također tipa "`string`", pri čemu su legalne vrijednosti "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K" i "A").

U slučaju da lista ne sadrži tačno 13 elemenata, funkcija treba baciti izuzetak tipa "`logic error`" uz prateći tekst "Neispravna lista!". U slučaju da se traži izbacivanje više karata od onog broja koliko ih zaista ima u špilu, postupak se prosto prekida kada špil ostane prazan (tj. izbacuje se onoliko karata koliko ih zaista ima, po opisanom postupku), bez prijave ikakve greške.

Funkcija "`VratiPosljednjihNKarata`" kao parametre prihvata referencu na špil (koji je, naravno, lista skupova), referencu na stek (istog onakvog tipa kakav vraća funkcija "`IzbaciKarte`"), kao i cjelobrojni parametar n, te da u listu špilova vrati posljednjih n karata u steku (tj. posljednjih n izbačenih karata). Ukoliko je broj n veći od broja elemenata steka, funkcija treba da baci izuzetak tipa "`range error`" uz prateći tekst "Nedovoljno karata u steku!". Ukoliko stek sadrži bilo kakve besmislene podatke (tj. podatke koji ne opisuju ni jednu smislenu kartu), funkcija treba da baci izuzetak tipa "`logic error`" uz prateći tekst "Neispravne karte!". Također, slično kao i kod prethodne funkcije, i ovdje treba baciti izuzetak tipa "`logic error`" uz prateći tekst "Neispravna lista!" ukoliko lista ne sadrži tačno 13 elemenata. U slučaju da se u steku nalazi neka karta koja se već nalazi u špilu, treba je prosto ignorirati.

Na kraju, funkcija "`PrebrojiKarte`" treba da kao parametar prihvati referencu na konstantni špil, a da kao rezultat (tipa "`int`") vrati koliko karata u tom špilu nedostaje (u odnosu na puni špil). Kao i kod prethodne dvije funkcije, treba baciti izuzetak tipa "`logic_error`" uz prateći tekst "Neispravna lista!" ukoliko lista ne sadrži tačno 13 elemenata.

Napisane funkcije istestirajte u glavnom programu u kojem se sa tastature najprije unosi korak razbrajanja za izbacivanje, te broj karata koje je potrebno izbaciti. Nakon što se izbace elementi iz liste, potrebno je ispisati koliko karata ima u špilu te sve karte u špilu, a zatim omogućiti korisniku da unese broj karata koje želi vratiti u špil, te ispisati broj karata koje su ostale u steku, kao i karte koje su ostale u steku.

Dijalog sa korisnikom treba izgledati ovako:

```
Unesite korak razbrajanja: 12
Unesite broj karata koje zelite izbaciti: 6
U spilu trenutno ima: 46 karata, i to:
Herc: 2 3 4 5 6 7 8 10 J Q A
Karo: 2 3 4 5 6 7 8 10 J K A
Pik:  2 3 4 5 6 7 8 9 10 Q K A
Tref: 2 3 4 5 6 7 8 9 J Q K A
Unesite broj karata koje zelite vratiti u spil: 2
Broj karata u steku: 4
Trenutne karte u steku:
Tref 10
Pik J
Karo Q
Herc K

----------------------------------------------------------------

Unesite korak razbrajanja: -5
Unesite broj karata koje zelite izbaciti: 3
Izuzetak: Neispravni elementi za izbacivanje!
```

Napomena: u svakom *catch*-bloku prije ispisivanja izuzetka treba biti ispisan tekst: "Izuzetak: "!

Napomena 2: Karte se ne smiju smještati ni u kakve pomoćne kontejnerske strukture (razne pomoćne vektore, liste, skupove, mape, itd.) osim onih koje su navedene u postavci zadatka. Eventualno je dozvoljeno nazive karata čuvati u nekom nizu ili vektoru.

4. Riješite ponovo prethodni zadatak, ali tako što ćete umjesto bibliotečki definiranog tipa podataka "`list`" koristiti ručno kreiranu povezanu listu čvorova (dakle, bez upotrebe ikakvih bibliotečki definiranih tipova), koji modeliraju karte u špilu. To ćete izvesti ovako. Prvo ćete definirati globalnu strukturu "`Karta`" koja sadrži polja "`boja`" (tipa "`Boje`") i "`vrijednost`" (tipa "`string`") koje sadrže respektivno boju i vrijednost karte (legalne vrijednosti su iste kao u prethodnom zadatku). Nakon toga definirajte još jednu globalnu čvornu strukturu "`Cvor`" koja opisuje jedan element liste. Ona treba sadržavati atribut "`karta`", koji je tipa "`Karta`", te "`sljedeci`", koji je po tipu pokazivač na objekat tipa "`Cvor`", a uloga mu je da pokazuje na sljedeći elemet liste. Funkcija "`KreirajSpil`" (bez parametara), koja vraća kao rezultat kompletan kreiran špil, u ovom zadatku treba da kreira povezanu listu karata (tačnije objekata tipa "`Cvor`") u kojoj se nalaze sve karte u špilu, tako što će prvo ići sve karte boje herc, redom po hijerarhiji (2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A), zatim sve karte bije karo (također redom po hijerarhiji), itd. Naravno, pri tome polje "`sljedeci`" u svakom od čvorova pokazivaće na sljedeći čvor, osim posljednjeg čvora, u kojem će to polje pokazivati ponovo na prvi čvor, čime se zapravo kreira krug karata (takve povezane liste u kojima posljednji čvor u listi pokazuje nazad na prvi čvor nazivaju se *kružne* ili *cirkularne liste*). Kao rezultat, funkcija "`KreirajSpil`" vraća pokazivač na prvi kreirani čvor (koji odgovara prvoj karti).

   Funkcija "`IzbaciKarte`" u ovom zadatku također prihvata dva parametra. Prvi je referenca na pokazivač na čvor koji odgovara prvoj karti u špilu, a drugi je, isto kao u prethodnom zadatku, referenca na konstantni uređeni par sa cjelobrojnim koordinatama koji opisuje šta treba izbaciti (sa istim značenjem kao u pethodnom zadatku). Funkcija kao rezultat vraća red (da, ovaj put je u pitanju red, odnosno kontejnerski tip "`queue`") uređenih parova koji će sadržavati izbačene karte, pri čemu ti parovi imaju isto značenje kao u prethodnom zadatku. Isto tako, u slučaku da par sadrži neispravne podatke, baca se izuzetak kao u prethodnom zadatku. Samo odstanjivanje obavlja se tako što se krećemo kružno kroz listu dok ne dođemo do karte koju treba izbaciti. Kada dođemo do nje, pokazivač "`sljedeci`" u čvoru koji prethodi čvoru na kojem se trenutno nalazimo (tj. koji odgovara karti koju izbacujemo) preusmjerava se tako da ne pokazuje više na čvor na kojem se trenutno nalazimo, nego na čvor koji slijedi iza njega (čime se karta efektivno odstranjuje iz špila). Tom prilikom, pomoću operatora "**`delete`**" potrebno je izbrisati čvor koji odgovara izbačenoj karti, da ne zauzima više memoriju., a podatke o izbačenoj karti (u vidu uređenog para) upisujemo u red. Postupak se ponavlja sve dok ne izbacimo onoliko karata koliko je traženo. U slučaju da se traži izbacivanje više karata od onog broja koliko ih zaista ima u špilu, postupak se prosto prekida kada špil ostane prazan (tj. izbacuje se onoliko karata koliko ih zaista ima, po opisanom postupku), bez prijave ikakve greške, isto kao u prethodnom zadatku. U slučaju da se lista potpuno isprazni, treba modificirati prvi parametar da postane nul-pokazivač (to je jedan od razloga zbog čega je prvi parametar referenca). Također, ukoliko dođe do brisanja čvora na koji je pokazivao prvi parametar (tj. brisanja prvog čvora u listi), treba ažurirati prvi parametar tako da pokazuje na sljedeći čvor, što obezbjeđuje da će nakon završetka funkcije

pokazivač koji odgovara prvom parametru biti uvijek validan. U svim ostalim slučajevima on ostaje netaknut. Za razliku od prethodnog zadatka, ovdje ne treba provjeravati da li je lista konzistentna, odnosno koliko se u njoj nalazi čvorova, i da li su u njima smislene informacije. Funkcija treba prosto da obavi opisani postupak šta god da se nalazilo u listi.

Umjesto funkcije "`VratiPosljednjihNKarata`", u ovom zadatku ćete implementirati funkciju "`VratiPrvihNKarata`" koja kao parametre prima referencu na pokazivač na čvor koji odgovara početku cirkularne liste (tj. prvoj karti u špilu), referencu na red (istog onakvog tipa kakav vraća funkcija "`IzbaciKarte`") koji sadrži karte koje treba vratiti nazad u špil, te cjelobrojni parametar n koji opisuje koliko karata treba vratiti. Ova funkcija treba da u špil (cirkularnu listu) vrati prvih n karata u redu (tj, prvih n izbačenih karata), svaku na svoje mjesto (ovdje ćete morati pomoću operatora "**`new`**" kreirati novi čvor koji odgovara karti koju ubacujete nazad u listu, nakon čega ćete se malo "poigrati" sa pokazivačima da uvežete novokreirani čvor u listu). Ukoliko je broj n veći od broja elemenata reda, funkcija treba da baci izuzetak tipa "`range error`" uz prateći tekst "Nedovoljno karata u redu!". Ukoliko red sadrži bilo kakve besmislene podatke (tj. podatke koji ne opisuju ni jednu smislenu kartu), funkcija treba da baci izuzetak tipa "`logic error`" uz prateći tekst "Neispravne karte!". Pri tome, radi jednostavnosti, pretpostavite da lista zaista sadrži konzistentne podatke (inače bi se stvari mogle znatno zakomplicirati). U slučaju da se u redu nalazi neka karta koja se već nalazi u špilu, treba je prosto ignorirati. Predvidite i mogućnost da prvi parametar može biti i nul-pokazivač (koji odgovara praznom špilu). U tom slučaju karte se vraćaju nazad u prazan špil, čime se efektivno kreira novi špil, a pokazivač na početak novokreiranog špila se smješta u prvi parametar.

Funkcija "`PrebrojiKarte`" kao parametar prihvata pokazivač na čvor koji odgovara prvoj karti u špilu (ili nul-pokazivač u slučaju praznog špila), a vraća koliko karata u tom špilu nedostaje (u odnosu na puni špil), pri čemu ćemo također pretpostaviti da lista sadrži konzistentne podatke. Konačno, treba implementirati i funkciju "`UnistiSpil`" koja kao parametar prihvata pokazivač na čvor koji odgovara prvoj karti u špilu (ili nul-pokazivač), a koja uklanja sve karte iz špila (ne smještajući pri tome nigdje informacije o njima), čime efektivno oslobađa memoriju koju su one zauzimale.

Napisane funkcije istestirajte u glavnom programu u kojem se sa tastature najprije unosi korak razbrajanja za izbacivanje, te broj karata koje je potrebno izbaciti. Nakon što se izbace elementi iz liste, potrebno je ispisati koliko karata ima u špilu te sve karte u špilu, a zatim omogućiti korisniku da unese broj karata koje želi vratiti u špil, te ispisati broj karata koje su ostale u redu, kao i karte koje su ostale u redu. Na kraju programa treba uništiti špil, tj. osloboditi memoriju koju je zauzimao. Dijalog sa korisnikom je isti kao u prethodnom zadatku, samo što umjesto "steku" treba da piše "redu".

Napomena: U zadatku nije dozvoljeno koristiti nikakve pomoćne bibliotečke kontejnerske strukture (vektore, liste, skupove, itd.) niti dinamički alocirane nizove. Eventualno je dozvoljeno nazive karata čuvati u nekom nizu ili vektoru.

5. Riješite ponovo prethodni zadatak, ali tako što će će se svugdje umjesto običnih koristiti pametni pokazivači (ovo uključuje i parametre funkcija koji su pokazivači). Suštinska razlika je što Vam ovaj put neće trebati operator "**`delete`**", jer će svaki čvor automatski nestati čim se isključi iz lanca, s obzirom da tada niko neće pokazivati na njega. Međutim, morate paziti da kada ostane samo jedan čvor, njegovo polje "`sljedeci`" će pokazivati na njega samog (tj. upravo na taj jedini preostali čvor). Ovu vezu *obavezno morate raskinuti* da biste obrisali taj posljednji čvor, inače će doći do curenja memorije (jer će taj čvor nastaviti da "čuva samog sebe" čak i kad nestanu drugi pokazivači koji su na njega pokazivali). U ovom zadatku nećete imati funkciju "`UnistiSpil`". Međutim, trebate se pobrinuti da po izlasku iz programa nemate curenje memorije koje se ovdje može pojaviti bez obzira što se koriste pametni pokazivači (zbog činjenice da pokazivači u kružnoj listi "čuvaju jedan drugog" čak i kad niko drugi ne pokazuje na ijedan čvor). Jednostavno rečeno, kružnu listu *morate na nekom mjestu raskinuti prije nego što program završi sa radom*.