Zadatak 1

Napišite parametriziranu strukturu `spoji` koja predstavlja ploču od 6 redaka i 7 stupaca na kojoj se igra modificirana verzija igre *Connect Four*, u kojoj dva igrača naizmjence stavljaju svoje pločice i svakome je cilj napraviti horizontalni niz od 4 uzastopne pločice. Struktura treba imati sljedeće članice:

- konstruktor: prima dva parametra koji označavaju oznaku za svakog igrača (onog tipa po kojem je struktura parametrizirana i kojeg `cout` zna ispisati).
- `ubaci` – prima indeks `i` $(1 \leq i \leq 7)$. U i-ti stupac odozgora ubacuje pločicu onog igrača koji je na redu. Pločica pada na najdonje slobodno mjesto u tom stupcu. Funkcija vraća 1 ako je element uspješno ubačen i u sljedećem koraku je na redu sljedeći igrač. Ako je taj stupac već bio popunjen, funkcija vraća 0 i isti igrač ostaje na redu u sljedećem koraku.
- `pobijedio` – vraća 1 ako je stanje na ploči takvo da je pobijedio prvi igrač, 2 ako je pobijedio drugi igrač, a inače 0. Možete pretpostaviti da će uvijek biti najviše jedan pobjednik.
- `ispis` – ispisuje ploču, i to svaki redak ploče u odvojeni redak na ekranu. Sve elemente jednog retka ispisati jedan iza drugog, a ako na nekom mjestu nema pločice, ispisati znak `'.'`, točno kao u primjeru niže.
- destruktor ispisuje konačno stanje na ploči. Također treba ispisati poruku je li pobijedio prvi ili drugi igrač ili nema pobjednika.

Primjer:

```
int main() {

spoji<char> A(4,'*','o');

A.ubaci(1);   A.ubaci(2);   A.ubaci(2);   A.ubaci(2);
A.ubaci(3);   A.ubaci(5);   A.ubaci(4);   A.ubaci(4);
A.ubaci(3);   A.ubaci(4);   A.ubaci(4);   A.ubaci(3);
A.ubaci(5);

A.ispis();

A.ubaci(4); A.ubaci(4); A.ubaci(4); A.ubaci(5);

return 0;

}
```

Ispis#1

```
.......
.......
...*...
.ooo...
.**o*..
*o**o..
```

Ispis#2

```
...*...
...o...
...*...
.oooo..
.**o*..
*o**o..

Pobijedio je drugi igrac.
```

U rješenju ne smijete koristiti STL-spremnike, a rješenje spremite pod imenom `zadatak1.cpp`.

U istoj datoteci napišite i `main` za testiranje ove strukture sa tipom `int`.

Zadatak 2

Na natjecanju *Eurosong* natječu se države iz cijele Europe. Svaka od tih država ocjenjuje ostale države. Zaduženi ste za tehničku podršku na sljedećem *Eurosongu* pa morate napisati program koji vodi bodovnu evidenciju.

Evidencija bodova je preslikavanje koje državi (`string`) pridružuje vektor uređenih parova koji predstavljaju države za koje je ta država glasala. Na prvom mjestu u paru nalazi se ime države (`string`), a na drugom broj bodova (`int`) koji je dana država dobila. Država koja glasa daje bodove od 1 do 12, ne mora dati sve bodove, ali uvijek mora dati 12 bodova.

(a) Napišite glavni program u kojem ćete učitati podatke o bodovima u varijablu `B`. Podaci će biti unošeni u formatu kao u donjem primjeru. Imena država sastoje se samo od velikih i malih slova bez razmaka. Prva država u svakom retku je država koja glasa. Države za koje se glasa će u svakom retku biti poredane prema broju bodova koje su dobile.

(b) U glavnom programu ispišite državu koja ima najviše bodova i broj bodova koji je sakupila. Možete pretpostaviti da neće biti više država s istim brojem bodova na prvom mjestu. Smijete koristiti pomoćne *STL-containere* (vector, list, map, …), ali ne i pomoćna polja!

Primjer:

Ulazni podaci	Izlazni podaci

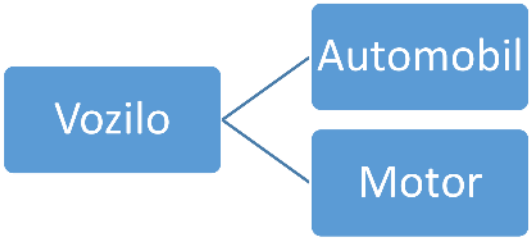
`Alb Nje 2 Fra 6 Hrv 12`
`Hrv Nje 1 Slo 3 BiH 5 Alb 12`
`BiH Slo 3 Fra 6 Hrv 10 Nje 12`
`Nje Hrv 2 BiH 4 Alb 10 Fra 12`
`Fra Nje 4 BiH 8 Alb 12`
`Slo Alb 3 Nje 6 BiH 12`
`kraj`
	`Alb 37`

Za učitavanje i ispisivanje koristite isključivo `cin` i `cout`.

Cjelokupni program spremite pod imenom `zadatak2.cpp`.

Zadatak 3

Zadana je sljedeća hijerarhija klasa:



Klasa `Vozilo` je apstraktna klasa koja reprezentira vozilo. Vozilo sadrži sljedeće varijable članice: `registracija` (`string`), `tezina` (`int`), `brzina` (`double`) i oznaku je li u prometu ili isključeno iz prometa. Dodatno, automobili imaju parametar `snaga (int)`, a motori `zapremnina (int)`.

Implementirajte sljedeće funkcije u danim klasama:

- konstruktor: označava da je vozilo u prometu i postavlja ostale varijable na vrijednosti koje su primljene kao parametri.
- `natovari(double t)`: povećava težinu za `t`
- `ubrzaj(double v)`: povećava brzinu vozila, i to za `min(v, snaga/100)` za `Automobil`, a za `Motor` za `min(v, zapremnina/50)`.
- `uspori(double v)`: smanjuje brzinu vozila, prema istoj formuli kao kod ubrzanja.
- `sudar(Vozilo &V)`: postavlja brzinu oba vozila koja sudjeluju u sudaru na 0. Ono vozilo koje ima manju težinu je isključeno iz prometa. Neka je A vozilo koje poziva funkciju. Ako je A motor, onda je A isključen iz prometa. Funkcija vraća referencu na A.
- `iskljuciIzPrometa`: funkcija koja isključuje iz prometa sva vozila koja voze brže od brzine vozila koje poziva funkciju.
- `ispisiIskljucena`: statička funkcija koja ispisuje registarske oznake svih vozila koja su isključena iz prometa.
- destruktor: ispisuje `"Unisten Automobil/Motor"`, ovisno o tome koji objekt ga je pozvao.

Napišite main koji testira ove funkcije. U njemu kreirajte jedno polje koje sadrži barem 6 različitih vozila (3 automobila i 3 motora). Pozovite svaku od gore navedenih funkcija. Ispišite na ekran stanje (naziv i sadržaj promjenjivih varijabli članica) na početku i na kraju.

Zadatak spremite pod imenom **zadatak1.cpp**

Zadatak 4

Implementirajte klasu `Broj`, čiji objekti predstavljaju prirodne brojeve zapisane u nekoj bazi. Broj i baza neka budu tipa `int`. Klasa treba imati sljedeće konstruktore i operatore:

- `Broj(s, b)`: stvara broj `n` u bazi `b` ($2 \leq b \leq 36$). Broj `n` je zadan kao string `s` koji se sastoji od znamenki tog broja, a svaka znamenka je broj između `0` i `9` ili veliko slovo od `A` do `Z`
- `Broj(x, b)`: stvara broj `n` u bazi `b` ($2 \leq b \leq 36$). Parametar `x` označava broj `n` u bazi 10
- omogućiti poziv konstruktora s jednim ili dva argumenta. Jedan argument stvara broj u bazi 10
- defaultni konstruktor stvara broj 1 u bazi 10
- `cout<<N`: ispis, u obliku "$a_n a_{n-1} \ldots a_1 a_0(b)$", gdje su $a_n, a_{n-1}, \ldots, a_1, a_0$ znamenke broja `N` u bazi `b`, npr. broj 161 u bazi 16 bio bi zapisan kao "`A1(16)`"
- `M+N, M*N` : uobičajeno zbrajanje i množenje brojeva, rezultat je objekt klase `Broj` u bazi koja je najmanji zajednički višekratnik baza od `M` i `N`. Možete pretpostaviti da će i nova baza biti manja ili jednaka od 36
- `N++`: povećava broj `N` za 1
- `N--`: smanjuje broj `N` za 1 ako je `N>0`, inače `N` ostaje nepromijenjen
- `++N`: povećava bazu za 1 ako je manja od 36
- `--N`: smanjuje bazu za 1 ako je veća od 2

kod operatora `++` i `--` (postfix i prefix), vraća se broj `N` nakon povećanja ili smanjenja

- konverzija u `int`: vraća dani broj pretvoren u bazu 10 kao `int`
- `N[int b]`: vraća `string` koji predstavlja `N` zapisan u bazi `b`. Ako je `b==0`, onda vraća string koji predstavlja `N` zapisan u svojoj bazi.
- `N^k`: računa broj `N` na potenciju `k (int)`, vraća objekt klase `Broj` u istoj bazi kao `N`. Možete pretpostaviti da je $k \geq 1$.

Ne morate pisati const kvalifikatore (umjesto const& možete prenositi po vrijednosti), i sve može biti public (dakle, ne trebate ni friend deklaracije). Omogućite ulančavanje gdje god je to moguće. Napišite i neki main koji testira bar 5 gornjih operatora.

Zadatak 5

Zadana je sljedeća hijerarhija klasa:

Klase predstavljaju jednodimenzionalan teren u kojem se nalaze planine (iznad površine mora) i podmorje (ispod površine mora). Reprezentirane su nizom od konačno mnogo realnih brojeva, pri čemu pozitivan broj opisuje visinu planine, dok negativan broj opisuje dubinu podmorja.
Klasa SamoGore je teren u kojem nema podmorja (svi brojevi u tom nizu su nužno nenegativni). Klasa Brijeg predstavlja poseban oblik klase SamoGore duljine tri, u kojem je visina srednjeg dijela planina veća od druga dva (rubna) dijela. Nijedna klasa nije apstraktna.

Bitne napomene – dobro pročitati:
Funkcije navedene u nastavku su članske i zajedničke su za sve klase (ako nije drugačije navedeno). Implementirajte ih bez dupliciranja kodova. Također, za svaku podklasu napravite i jednostavniju implementaciju, ako ona postoji. Dozvoljeno je dopisati i pomoćne funkcije.
Sakrijte sav sadržaj iz klasa prema van, uključujući i sve članske varijable svih klasa.
Na kraju napišite *main* koji testira sve funkcije.

konstruktor – Zajednički za sve klase. Prima vektor `doubleova` proizvoljne duljine, koji predstavlja visine (i dubine) odgovarajućih dijelova terena. Dodatno, neka postoji konstruktor za klasu `Brijeg` koji prima jedan nenegativan `double x` te stvara brijeg s visinama `x/2,x,x/3` redom.

`globalno_zatopljenje` – Prima jedan pozitivan `double x`. Uslijed globalnog zatopljenja, razina mora raste. Zato se svaki komad terena u nizu visina i dubina umanjuje za `x`. U slučaju objekata `SamoGore` i `Brijeg`, ako je `x` veći od visine najmanje uzvisine, tada se umjesto za `x`, svaki komad terena umanjuje za visinu te najmanje uzvisine.

`potres` – Ne prima ništa. Na kraju funkcije, teren je sada sortiran silazno (počevši od najveće uzvisine, pa do najdubljeg dijela podmorja). Ne radi ništa ako je funkcija pozvana na objektu tipa `Brijeg`.

`ispis` – Ispisuje sve visine (i dubine) redom na ekran.

destruktor – Ispisuje `„Unisteno GoreIMore“/„Unistene SamoGore“/„Unisten Brijeg“` prilikom uništenja objekta.

`spoji_sve` – Globalna funkcija koja ne prima niti ne vraća ništa. Stvara novi objekt tipa `SamoGore` koji se dobije spajanjem svih `Brijegova` iz programa. `Brijegove` možete spojiti bilo kojim redoslijedom. Ukupna duljina planine reprezentirane objektom `SamoGore` je `3*n`, gdje je `n` broj svih `Brijegova`. Oprez: neka se prilikom poziva ove funkcije u glavnom programu ne pozove destruktor ni na kojem objektu; pazite na to da su elementi svih klasa skriveni van svih klasa.

Zadatak spremite pod imenom **`zadatak1.cpp`**

Zadatak 6

Zadana je sljedeća hijerarhija klasa:

Svaka klasa opisana je konačnim nizom cijelih brojeva različitih od nule. Ukoliko je koeficijent na `i`-tom mjestu jednak `x`, `k` kuna koje imamo u budžetu pretvaramo u `k*x` kuna (ako je `x` pozitivan), odnosno u `max(k-|x|,0)` kuna (ako je `x` negativan). Čovjek igrajući lutriju ulazi s izvjesnom količinom novca u budžetu, te primjenjuje oklade redom, počevši od prve prema zadnjoj, u svakom koraku s novom količinom novca u budžetu.
`JednakaLutrija` ima svojstvo da je to niz od parno mnogo elemenata takvih da je zbroj prva dva elementa niza jednak nuli, zbroj trećeg i četvrtog jednak nuli, (…), zbroj zadnjeg i predzadnjeg jednak nuli. `Listic` je `JednakaLutrija` duljine 2. Nijedna klasa nije apstraktna.

Bitne napomene – dobro pročitati:
Funkcije navedene u nastavku su članske i zajedničke su za sve klase (ako nije drugačije navedeno). Implementirajte ih bez dupliciranja kodova. Također, za svaku podklasu napravite i jednostavniju implementaciju, ako ona postoji. Dozvoljeno je dopisati i pomoćne funkcije.
Sakrijte sav sadržaj iz klasa prema van, uključujući i sve članske varijable svih klasa.
Na kraju napišite *main* koji testira sve funkcije.

konstruktor – Prima vektor `doubleova` proizvoljne duljine, koji predstavlja koeficijente redom u lutriji. Smijete pretpostaviti da će taj vektor biti valjan u slučaju objekata klase `JednakaLutrija`. Dodatno, neka postoji konstruktor za klasu `Listic` koji prima jedan `double x` različit od nule te stvara listić s koeficijentima `x,-x` redom.

zamijeni_jednu – Prima jedan `int x` različit od nule, te jedan `int i`. Mijenja `i`-ti koeficijent tako da sada na tom mjestu piše `x`. U slučaju klasa `JednakaLutrija` i `Listic` izmijenite i odgovarajući susjedni koeficijent tako da svojstvo klase `JednakaLutrija` bude očuvano (primjerice, ako je pozvana funkcija s `i=2`, treba promijeniti i prvi element na `-x`).

izračunaj_iznos – Za primljeni `double k`, vraća `double y` koji je jednak iznosu u budžetu nakon cijelog niza oklada na objektu tipa `Lutrija/JednakaLutrija/Listic`.

`ispis` – Ispisuje sve koeficijente redom na ekran.

destruktor – Ispisuje „`Unistena Lutrija`"/„`Unisten JednakaLutrija`"/„`Unisten Listic`" prilikom uništenja objekta.

`spoji_listice` – Globalna funkcija koja ne prima niti ne vraća ništa. Stvara novi objekt tipa `JednakaLutrija` koji se dobije spajanjem svih `Listica` iz programa. `Listice` možete spojiti bilo kojim redoslijedom. Ukupna duljina lutrije reprezentirane objektom `JednakaLutrija` je `2*n`, gdje je `n` broj svih `Listica`. Oprez: neka se prilikom poziva ove funkcije u glavnom programu ne pozove destruktor ni na kojem objektu; pazite na to da su elementi svih klasa skriveni van svih klasa.

Zadatak spremite pod imenom **`zadatak1.cpp`**

Zadatak 7

Zadana je sljedeća hijerarhija klasa:

Klase opisuju ploču Kola sreće, na kojoj se nalazi jedna riječ sastavljana od velikih slova engleske abecede, u kojoj moguće nedostaju neka slova (na tim mjestima nalazi se malo englesko slovo `'x'`). Klasa `Palindrom` opisuje ploče Kola sreće na kojima se konačno rješenje jednako čita slijeva na desno i zdesna na lijevo. Primjerice „`ABCxx`" je jedna dozvoljena ploča klase `Palindrom` (jer se zadnja dva slova mogu postaviti na `B` i `A` redom), „`AxxB`" nije dozvoljena ploča klase `Palindrom`. Klasa `Trijec` je predstavlja ploče kao u klasi `Palindrom` koje su duljine tri. Nijedna klasa nije apstraktna.

Bitne napomene – dobro pročitati:
Funkcije navedene u nastavku su članske i zajedničke su za sve klase. Implementirajte ih bez dupliciranja kodova. Također, za svaku podklasu napravite i jednostavniju implementaciju, ako ona postoji. Dozvoljeno je dopisati i pomoćne funkcije.
Neka sve članske varijable svih klasa budu skrivene van klasa, te ako je moguće i svojih podklasa, Na kraju napišite *main* koji testira sve funkcije.

konstruktor – Prima (`c++`-ovski) string te stvara ploču Kola sreće, za svaku od klasa. Možete pretpostaviti da će taj string biti valjan ukoliko je pozvan na konstruktoru za `Palindrom` i `Trijec`.

`dodaj_slovo` – Prima jedan `char c`, i jedan `int i`. Postavlja `char c` na mjesto `i` na ploču Kola sreće, u svakoj od klasa, ukoliko je to mjesto slobodno (dakle ako se tamo nalazi znak `'x'`), inače ne radi ništa. Funkcija ne provjerava hoće li svojstvo `Palindroma` ili `Trijeci` biti očuvano, ako se funkcija pozove na tom objektu.

`dozvoljen` – Ne prima ništa. Ukoliko je pozvana na objektu tipa `KoloSrece`, vraća `true`. Inače provjerava zadovoljava li ploča u objektu koji je pozvao funkciju svojstvo klase `Palindrom` (tj. mogu li se praznine `'x'` i dalje popuniti slovima tako da riječ na ploči na kraju bude palindrom). Ako je zadovoljen uvjet, vraća `true`. Ako nije, vraća `false`, te stvara novi objekt tipa `KoloSrece` kojem je ploča jednaka ploči objekta koji je pozvao funkciju. Objekt koji je pozvao funkciju se ne uništava. Oprez: neka se prilikom poziva ove funkcije u glavnom programu ne pozove destruktor ni na kojem objektu.

destruktor – Ispisuje „`Unisteno KoloSrece`"/„`Unisten Palindrom`"/„`Unistena Trijec`" prilikom uništenja objekta.

`koliko` - Globalna funkcija koja ne prima ništa. Ispisuje prve dvije trećine ploča svih objekata `Trijeci` na ekran, bilo kojim redoslijedom. Oprez: pazite na to da su elementi svih klasa skriveni van svih klasa.

Zadatak spremite pod imenom **`zadatak1.cpp`**

Zadatak 8

Programirate igru u kojoj se igrač bori raznim oružjem protiv neprijatelja. Napišite klasu (ili strukturu, po izboru; u nastavku će svuda pisati riječ klasa) `Predmet` koja predstavlja baznu klasu za predmete u igri. Ima samo jednu `protected` varijablu `izdrzljivost` (`int`) te jedan konstruktor koji prima izdržljivost kao argument. Zatim napišite klase `Oklop` i `Oruzje` koje nasljeđuju `Predmet` te klasu `Mac` koja nasljeđuje `Oruzje`. U **jednoj** navedenoj klasi možete **jednu** klasu označiti sa `friend`.

Klasa `Oklop` ima:

- `protected` varijablu `obrana` (`int`) te
- konstruktor koji prima izdržljivost i obranu kao argumente.

Klasa `Oruzje` ima

- `protected` varijable `preciznost` (`int`) i `snaga` (`int`),
- javnu čistu virtualnu funkciju `list<int> upotrijebi(Oklop &o, int dozvoljeno_pokusaja)` te
- konstruktor koji prima izdržljivost, preciznost i snagu.

Klasa `Mac` ima

- javnu statičku varijablu `kriticnih_udaraca` (`int`, inicijaliziran na 0) i
- jedan konstruktor (prima isto što i konstruktor za `Oruzje`)

te implementira funkciju `upotrijebi` ovako: dok mač ili oklop ne budu uništeni (tj. izdržljivost danog predmeta postane nepozitivna) ili broj dozvoljenih pokušaja prijeđen, mačem se pokuša udariti oklop. Generira se slučajan broj u skupu $\{1, 2, \ldots,$ `preciznost`$\}$ i ako je on veći ili jednak od vrijednosti obrane oklopa, onda je mač uspješno udario: maču se izdržljivost smanji za 1 te se oklopu izdržljivost smanji za snagu mača. Funkcija vraća listu sljedeća četiri broja: koliko je udaraca tim mačem pokušano, koliko je udaraca tim mačem bilo uspješno, završnu izdržljivost mača te završnu izdržljivost oklopa. Dodatno, varijabla `kriticnih_udaraca` broji sve udarce svih mačeva gdje je generirani slučajni broj bio jednak vrijednosti `preciznost`.

Napišite program koji za tri mača i jedan oklop isproba svaki mač na tom oklopu; preciznije, upotrijebi kopiju svakog mača na po jednoj kopiji oklopa. Program ispisuje rezultate isprobavanja u tri retka, po jedan za svaki mač. Svaki redak sadrži redom brojeve iz liste koje vrati funkcija `upotrijebi` (odvojene razmacima).

Čitavo rješenje spremite u datoteku `zadatak1.cpp`.

Nije dozvoljeno stavljati dodatne konstruktore, članske funkcije i varijable u navedene klase te argumente u definicije konstruktora i funkcija. Za generiranje slučajnih brojeva možete koristiti funkciju `std::rand` iz `<cstdlib>`.

Zadatak 9

Implementirajte strukture (ili klase, svi članovi smiju biti javni) `Zaba` i `Cesta` s (barem) dolje navedenim funkcijama i operatorima. Ako nije drugačije uvjetovano zadatkom, operatore možete implementirati kao funkcije članice ili kao globalne funkcije. Ako imate poteškoća s međuovisnim definicijama, na vrh dodajte deklaracije `struct Zaba;` `struct Cesta;` i operatore koji su funkcije članice definirajte ispod definicija struktura.
`Cesta` predstavlja cestu duljine d (cijeli broj, $d \geq 0$). `Zaba` predstavlja žabu koja putuje nekom cestom duljine d. `Zaba` pamti duljinu d ceste kojom putuje, te svoju trenutnu poziciju x (cijeli broj, $0 \leq x \leq d$). Zamišljamo da se cesta duljine d sastoji od d blokova, te žaba može biti ispred prvog ($x = 0$), između neka dva bloka, ili nakon posljednjeg bloka ($x = d$). `Zaba` ne pamti objekt/referencu/pokazivač na cestu kojom putuje, već samo njenu duljinu. Osim kretanja postojećom cestom, žabu je moguće teleportirati na bilo koju cestu, i to uvijek na početak te ceste (poziciju 0).

- Izraz `zaba++` (samo postfiksni inkrement) pomiče žabu za jedan korak unaprijed na trenutnoj cesti. Ako je žaba prije evaluacije izraza `zaba++` već na kraju trenutne ceste (ako $x = d$), žabina nova pozicija nakon poziva je 0. Izraz vraća uređeni par nove pozicije žabe (x) i ukupne duljine trenutne ceste (d).
- Izraz `zaba >> cesta` teleportira žabu na početak ceste `cesta`. Izraz neka ne vraća povratnu vrijednost.
- Poziv `cesta[n]` uzima pozitivan cijeli broj n i vraća novu cestu čija je duljina d/n (cjelobrojno dijeljenje).
- Poziv `*zaba` (unarni operator `*`) stvara cestu koja je točno tri puta dulja od ceste kojom se žaba trenutno kreće. Potom teleportira žabu na tu novu cestu, a povratna vrijednost izraza je nova cesta.
- Napišite globalnu funkciju `int zabalans(set<Zaba *>)`. Funkcija simulira proces koji u svakom koraku poziva `++zaba` za svaku žabu. Funkcija se treba prekinuti kad se prvi put ispuni sljedeći uvjet: na poziciji 0 je 0 žaba, a na svakoj preostaloj poziciji p (tj. za sve $p > 0$) najviše je 2^p žaba (duljina najdulje ceste, a time i p s kojim računamo, bit će najviše 30). Ako se ovakva konfiguracija nikada ne dogodi, funkcija se ne prekida (ulazi u beskonačnu petlju). Povratna vrijednost funkcije (onda kad se proces završava) broj je koraka simuliranog procesa nakon kojeg je uvjet ispunjen. Primjerice, ako imamo žabu a na poziciji 0 na cesti duljine 1, žabu b na poziciji 3 na cesti duljine 3, te žabu c na poziciji 4 na cesti duljine 4, te proslijedimo skup {`&a, &b, &c`}, funkcija će vratiti broj 3, te će a biti na poziciji 1, dok će b i c biti na poziciji 2.
- Pozivi oblika `cout << s` gdje je `s` tipa `set<Zaba *>` trebaju ispisati žabe i pripadne ceste (u bilo kojem poretku), i to tako da je svaka žaba u vlastitom retku. Žaba se ispisuje kao malo slovo `z` ako nijedna druga žaba u skupu nije na istoj poziciji, a inače se ispisuje kao veliko slovo `K`. Cesta je reprezentirana minusima (njih ukupno d), a simbol žabe ispisuje se na odgovarajućoj poziciji između minusa (ispred svih minusa ako $x = 0$ odnosno nakon svih minusa ako $x = d$). Primjerice, za skup `s = {&a, &b, &c}` iz prethodne točke, prije poziva funkcije `zabalans`, poziv `cout << s` daje sljedeći ispis:

  ```
  z-
  ---z
  ----z
  ```

  Nakon poziva funkcije `zabalans`, poziv `cout << s` daje sljedeći ispis:

  ```
  -z
  --K-
  --K--
  ```

  Pozive ovog operatora mora biti moguće ulančavati.
- Napišite i funkciju `main` u kojoj demonstrirate sve zadane funkcije i operatore.

Čitavo rješenje spremite u datoteku `zadatak2.cpp`.
Detalji koji nisu specificirani ostavljeni su vama na izbor. Smijete koristiti standardnu biblioteku i bilo koju funkcionalnost dostupnu u standardu C++11 neovisno o tome jesmo li je spomenuli na kolegiju, osim ključne riječi `auto`. Nigdje ne treba provjeravati smislenost argumenata operatora, npr. nikad se neće evaluirati izrazi poput `cesta[0]`. Nigdje nije nužno koristiti `const`.

Zadatak 10

Programirate igru u kojoj se igrač bori protiv raznih neprijatelja. Napišite klasu (ili strukturu, po izboru; u nastavku će svuda pisati riječ klasa) `ZivoBice` koja predstavlja baznu klasu za živa bića u igri. Ima samo `protected` varijablu `zdravlje` (`int`) te jedan konstruktor koji prima samo `zdravlje` kao argument. Zatim napišite klase `Placenik`, `LikIgraca` i `Neprijatelj` koje nasljeđuju od klase `ZivoBice` te klasu `Zmaj` koja nasljeđuje od `Neprijatelj`. U **dvije** od navedenih klasa možete po **jednu** klasu označiti sa `friend`.

Klasa `Placenik` ima

- `protected` varijablu `cijena` (`double`) te
- konstruktor koji prima zdravlje i cijenu.

Klasa `LikIgraca` ima

- `protected` varijablu `placenici` (`vector<Placenik*>`) te
- konstruktor koji prima zdravlje i plaćenike.

Klasa `Neprijatelj` ima

- `protected` varijablu `snaga` (`int`),
- javnu čistu virtualnu funkciju `void napadni(LikIgraca &lik)` te
- konstruktor koji prima zdravlje i snagu.

Klasa `Zmaj` ima

- jedan konstruktor (koji prima isto što i konstruktor za klasu `Neprijatelj`)

te implementira funkciju `napadni` ovako: dok ili zmaj ili svi plaćenici danog lika igrača (ako ih ima; inače borba odmah završava) ne budu poraženi (zdravlje im postane nepozitivno; poraženi plaćenici se odmah brišu iz `placenici`), zmaj pokušava udariti nultog plaćenika. I za zmaja i za nultog plaćenika se generira slučajan broj u skupu $\{1, 2, 3\}$; ako je prvi manji od drugog, od zdravlja plaćenika se oduzme snaga zmaja. Ako je drugi broj manji, od zdravlja zmaja se oduzme 1. Ako su isti, zmaj i plaćenik oštete jedan drugog (zmaj plaćenika za snagu, a plaćenik zmaja za 1). Pretpostavite da pri pozivu ove funkcije i zmaj i svi plaćenici imaju pozitivno zdravlje.

Funkcija `napadni` ispisuje rezultate borbe u onoliko redaka koliko zmaj porazi plaćenika (jedan za svakog plaćenika). Svaki redak sadrži pet brojeva odvojenih razmacima: redni broj (počevši od 1, ne ispisuje se točka), broj udaraca koje je zmaj pokušao zadati plaćeniku, broj pokušanih udaraca koji su bili uspješni za zmaja (oni kod kojih je zmajeva snaga bila oduzeta od zdravlja plaćenika), početno zdravlje plaćenika te cijena plaćenika.

Napišite program u kojem četveročlana družina, tj. lik igrača s troje plaćenika, biva napadnut od strane ljutitog zmaja (vrijednosti poput zdravlja, cijena, itd. zadajte sami).

Čitavo rješenje spremite u datoteku `zadatak1.cpp`.

Nije dozvoljeno stavljati dodatne konstruktore, članske funkcije i varijable u navedene klase te argumente u definicije konstruktora i funkcija. Za generiranje slučajnih brojeva možete koristiti funkciju `std::rand` iz `<cstdlib>`.

Zadatak 11

Implementirajte strukture (ili klase, svi članovi smiju biti javni) `Stranica` i `Knjiga` s (barem) dolje navedenim funkcijama i operatorima. Ako nije drugačije uvjetovano zadatkom, operatore možete implementirati kao funkcije članice ili kao globalne funkcije. Ako imate poteškoća s međuovisnim definicijama, na vrh dodajte deklaracije `struct Stranica; struct Knjiga;` i operatore koji su funkcije članice definirajte ispod definicija struktura. `Stranica` predstavlja stranicu na kojoj piše neki tekst t (`string`, možda prazan) i koja ima redni broj stranice r (prirodan broj, $r \geq 1$). `Knjiga` predstavlja kolekciju stranica (sami odaberite spremnik) koja može biti prazna. Knjiga ne mora sama pamtiti informaciju o poretku stranica, informacija o poretku već je zabilježena u pojedinoj stranici kao redni broj stranice. Tokom pisanja knjige može se dogoditi da za neke redne brojeve imamo više stranica (to su zapravo razni kandidati za finalnu verziju stranice s tim rednim brojem), te će u nekom trenutku za svaku stranicu trebati odabrati jednu od tih verzija. Zbog toga je dopušteno da u knjizi čuvamo više od jedne stranice s istim rednim brojem.

- Izraz `stranica < txt`, gdje je `txt` tipa `string`, postavlja tekst stranice na `txt`. Slično, `stranica < rbr`, gdje je `rbr` tipa `int`, postavlja redni broj stranice na `rbr`. Ove operatore treba moći proizvoljno ulančavati na način da npr. `stranica < rbr < txt` postavlja redni broj stranice, a potom i tekst stranice. Uputa: napišite dva operatora `<` (tj. dva preopterećenja tog operatora).
- Izraz oblika `knjiga << s` dodaje stranicu `s` u knjigu. Omogućite i izraze poput `knjiga << s1 << s2`.
- Izraz oblika `knjiga1 & knjiga2` ne mijenja postojeće knjige, već stvara i vraća novu knjigu čija je kolekcija stranica jednaka uniji kolekcija stranica knjiga `knjiga1` i `knjiga2` (pretpostavite da su te kolekcije disjunktne).
- Izraz `-knjiga` (unarni minus). Definirajmo prvo sljedeći kriterij usporedbe stranica: stranica `a` je manja od stranice `b` ako i samo ako: `a` ima manji redni broj stranice od `b`, ili je taj broj jednak ali je `a` tekstom leksikografski manja od `b`. Pretpostavite da neće biti stranica s jednakim brojem rednim brojem i jednakim tekstom. Ako je `s` stranica, njen indeks u ovako definiranom poretku označimo s `r(s)`. Izraz `-knjiga` treba redni broj svake stranice `s` zamijeniti s `r(s)`. Pritom indeksiranje treba početi od jedinice. Primjerice, ako u knjizi imamo stranicu s rednim brojem 3 i tekstom "x", stranicu s rednim brojem 5 i tekstom "b" te stranicu s rednim brojem 5 i tekstom "a", tada po izlazu iz funkcije imamo: stranicu s rednim brojem 1 i tekstom "x", stranicu s rednim brojem 2 i tekstom "a", te stranicu s rednim brojem 3 i tekstom "b". Izraz `-knjiga` ne treba imati povratnu vrijednost.
- Pozivi oblika `cout << s` gdje je `s` tipa `knjiga` ispisuje sve stranice trenutno pohranjene u knjizi (njihov tekst) u poretku određenom rednim brojevima zapisanim u samim stranicama. Između svake dvije stranice treba prijeći u novi red. Ako za neki redni broj nema stranice (a za neki veći redni broj postoji stranica), umjesto teksta na tom rednom broju ispišite "/". Ako za neki redni broj postoji više stranica, ispišite samo tekstom leksikografski prvu među tim stranicama. Primjerice, ako u knjizi imamo stranicu s rednim brojem 3 i tekstom "x", stranicu s rednim brojem 5 i tekstom "b" te stranicu s rednim brojem 5 i tekstom "a", tada je ispis:

  ```
  /
  /
  x
  /
  a
  ```

  Pozive ovog operatora mora biti moguće ulančavati.
- Napišite i funkciju `main` u kojoj demonstrirate sve zadane funkcije i operatore.

Čitavo rješenje spremite u datoteku `zadatak2.cpp`.
Detalji koji nisu specificirani ostavljeni su vama na izbor. Smijete koristiti standardnu biblioteku i bilo koju funkcionalnost dostupnu u standardu C++11 neovisno o tome jesmo li je spomenuli na kolegiju, osim ključne riječi `auto`. Nigdje ne treba provjeravati smislenost argumenata operatora, npr. nikad se neće evaluirati izrazi poput `knjiga(2)`. Nigdje nije nužno koristiti `const`.

Zadatak 12

Programirate igru u kojoj igrač obavlja različite zadaće (eng. *quest*) kako bi osvojio bodove iskustva (eng. *experience points*, ili kratko *xp*). Napišite klasu (ili strukturu, po izboru; u nastavku će svuda pisati riječ klasa) `Zadaca` koja predstavlja baznu klasu za sve zadaće u igri. Ima jednu `protected` varijablu, `xp` (`int`), jednu javnu statičku varijablu `ukupni_xp` (`int`, inicijaliziran na nulu, vodi računa o ukupnom xp-u koji je igrač skupio u svim riješenim zadaćama) te jedan konstruktor koji prima samo `xp` kao argument. Zatim napišite klase `Neprijatelj` i `MiroljubivaZadaca` koje nasljeđuju klasu `Zadaca` te klasu `Dijalog` koja nasljeđuje klasu `MiroljubivaZadaca`. U **jednoj** navedenoj klasi možete **jednu** klasu označiti sa `friend`.

Klasa `Neprijatelj` ima

- `protected` varijablu `koef_ozljede` (`int`) te
- konstruktor koji prima argumente `xp` i `koef_ozljede`.

Klasa `MiroljubivaZadaca` ima

- javnu čistu virtualnu funkciju `void rijesi(Neprijatelj &n)`,
- `protected` varijablu `koef_uspjeha` (`int`) te
- konstruktor koji prima xp i koeficijent uspjeha.

Klasa `Dijalog` ima

- jedan konstruktor (koji prima isto što i konstruktor za klasu `MiroljubivaZadaca`) te

implementira funkciju `rijesi` ovako: s vjerojatnošću 1.0/`koef_uspjeha`, dani neprijatelj biva poražen mirnim putem, tj. dijalogom (nije ga uopće bilo potrebno napasti); u tom slučaju igrač osvaja xp za neprijatelja i za dijalog. U protivnom, igrač počinje s napadima: u svakom napadu, s vjerojatnošću 1.0/`koef_uspjeha` neprijatelj biva poražen i igrač osvaja samo xp za neprijatelja. Ako neprijatelj nije poražen u danom napadu, s vjerojatnošću 1.0/`koef_ozljede` igrač dobiva jednu ozljedu i kreće idući napad (primijetite da igrač ne dobiva ozljedu za neuspjeh u pokušaju dijaloga!). Naime, neprijatelja se napada sve dok ne bude poražen.

Funkcija `rijesi` ispisuje jedan redak koji sadrži četiri podatka odvojenih razmacima: broj napada koji je bio potreban da bi se zadaća riješila (iznosi 0 ako je riješena dijalogom), koliko ozljeda je igrač tijekom rješavanja dane zadaće primio od danog neprijatelja, koliko xp-a je igrač dobio za danog neprijatelja te zbroj xp-a neprijatelja i dijaloga (odnosno maksimalni xp koji se mogao osvojiti).

Napišite program u kojem se funkcija `rijesi` poziva na nekom neprijatelju. Pritom neka `koef_uspjeha` bude postavljen na 10, a ostale vrijednosti zadajte sami.

Čitavo rješenje spremite u datoteku `zadatak1.cpp`.

Nije dozvoljeno stavljati dodatne konstruktore, članske funkcije i varijable u navedene klase te argumente u definicije konstruktora i funkcija. Za generiranje slučajnih brojeva možete koristiti funkciju `std::rand` iz `<cstdlib>`.

Zadatak 13

Implementirajte strukture (ili klase, svi članovi smiju biti javni) `Knjiga` i `Autor` s (barem) dolje navedenim funkcijama i operatorima. Ako nije drugačije uvjetovano zadatkom, operatore možete implementirati kao funkcije članice ili kao globalne funkcije. Ako imate poteškoća s međuovisnim definicijama, na vrh dodajte deklaracije `struct Knjiga; struct Autor;` i operatore koji su funkcije članice definirajte ispod definicija struktura.
`Autor` predstavlja autora čiji je OIB `o` (`string`, možda prazan). Kad kažemo da je netko "autor knjige", ne pretpostavljamo da je to jedini autor te knjige. Drugim riječima, "autor knjige" uvijek znači "jedan od autora knjige (možda jedini)". `Knjiga` predstavlja knjigu o kojoj pamtimo samo autore `a`. To je kolekcija stringova (sami odaberite spremnik) čiji je svaki element OIB jednog od autora knjige. Nema nikakvih ograničenja na ovu kolekciju niti njen sadržaj (kolekcija može biti prazna, vrijednosti mogu biti prazne, vrijednosti se mogu ponavljati itd.). Knjiga ne pamti objekte/reference/pokazivače na autore, već samo vrijednosti njihovih OIB-a.

- Izraz `autor(knjiga)` vraća logičku vrijednost: je li `autor` autor knjige `knjiga`. Jednako se treba ponašati i izraz `knjiga(autor)`.
- Izraz oblika `knjiga % autor` dodaje autora `autor` među autore knjige `knjiga` ako taj autor već nije u njenoj kolekciji autora. Omogućite i izraze poput `knjiga % a1 % a2`.
- Izraz oblika `*knjiga` (unarni operator `*`) vraća OIB nekog od autora te knjige (sami odaberite kojeg) kao referencu na konstantan `string`. Ovakav izraz nikad se neće pozvati na knjigama bez autora.
- Izraz `knjiga->x` treba se evaluirati jednako kao izraz `OIB.x` ako je `OIB` referenca na konstantan `string` koju vraća izraz `*knjiga` iz prethodne točke. Primjerice, `size` je jedna `const` funkcija definirana za objekte tipa `string` pa `knjiga->size()` vraća istu vrijednost kao `(*knjiga).size()`. Uputa: povratni tip operatora `->` mora biti `const string*`.
- Poziv oblika `*vektor_knjiga` (unarni operator `*`) uzima objekt tipa `vector<Knjiga>` (elementi će biti jedinstveni), i vraća objekt tipa `map<string, double>`. Povratna vrijednost svakom autoru `a` koji je autor barem jedne knjige u `vektor_knjiga` pridružuje *broj bodova* $M + \frac{K}{2}$. Ovdje je M broj knjiga iz `vektor_knjiga` čiji je `a` autor, dok je K broj knjiga u `vektor_knjiga` za koje vrijedi da postoji neki autor `b`, `a` $\neq$ `b`, takav da je u barem nekoj knjizi iz `vektor_knjiga` bio autor zajedno s `a`.

  Pogledajmo primjer za knjige `k1` autora `A` i `B`, `k2` autora `A`, te `k3` autora `A` i `C`. `A` je autor tri knjige pa $M = 3$. Za izračunati K, uočimo da su zajedno s `A` autori nekih knjiga (knjiga `k1` i `k3`) bili `B` i `C`. Dakle, za izračunati K prebrojimo sve knjige kojima je barem jedan autor član skupa {`B, C`}. Dvije su takve knjige, pa $K = 2$. Stoga se autoru `A` pridružuje $3 + \frac{2}{2} = 4$.

  Izračunajmo taj broj i za autora `B`. On je autor jedne knjige pa $M = 1$. Za izračunati K, uočimo da je autor barem jedne knjige (i to knjige `k1`) uz `B` bio `A`. To je ujedno i jedini autor koji je autor neke knjige zajedno s `B`. Dakle, za izračunati K prebrojimo sve knjige kojima je autor `A`. Tri su takve knjige, pa $K = 3$. Stoga se autoru `B` pridružuje $1 + \frac{3}{2} = 2.5$. Za autora `C` račun je analogan (zamijenite uloge od `B` i `C`, te `k1` i `k2`).
- Pozivi oblika `cout << bodovi` gdje je `bodovi` tipa `map<string, double>` ispisuje sve parove, svaki u svom retku, redom tekst i broj (odvojene razmakom). Poredak može biti proizvoljan. Pozive ovog operatora mora biti moguće ulančavati.
- Napišite i funkciju `main` u kojoj demonstrirate sve zadane funkcije i operatore.

Čitavo rješenje spremite u datoteku `zadatak2.cpp`.
Detalji koji nisu specificirani ostavljeni su vama na izbor. Smijete koristiti standardnu biblioteku i bilo koju funkcionalnost dostupnu u standardu C++11 neovisno o tome jesmo li je spomenuli na kolegiju, osim ključne riječi `auto`. Nigdje ne treba provjeravati smislenost argumenata operatora, npr. nikad se neće evaluirati izrazi poput `knjiga(2)`. Nigdje, osim tamo gdje je eksplicirano, nije nužno koristiti `const`.

Zadatak 14

Implementirajte klasu **`Magneti`** koja reprezentira niz nula ili više horizontalno položenih magneta. Svaki magnet označavamo sa stringom duljine 4: `[+−]` ili `[−+]`, ovisno o tome koji mu je pol na lijevom, a koji na desnom kraju.

Poznato je da se različiti polovi međusobno privlače, a isti odbijaju. Zato definiramo **legalan niz magneta** kao niz magneta i praznina (reprezentiran stringom `[ ]` duljine 3) koji zadovoljava sljedeće:

- Između dva jednako usmjerena magneta nema praznina;
- Između dva različito usmjerena magneta postoji točno jedna praznina;
- Praznine se mogu pojavljivati samo između dva magneta (ne mogu se pojavljivati lijevo od najlijevijeg ili desno od najdesnijeg magneta).

Primijetite da za svaki niz magneta postoji točno jedan legalan niz.

Recimo, za niz od od tri magneta `[+−]`, pa dva magneta `[−+]`, pa opet jednog magneta `[+−]`, jedini legalan niz je: `[+−],[+−],[+−],[ ],[−+],[−+],[ ],[+−]`.

Popis i opis funkcija:

- *Konstruktori*
  Napravite (barem) dva konstruktora. Prvi konstruktor **`int`** `n` i **`string`** `s`, gdje je `s` jednak `[+-]` ili `[-+]` te stvara niz istih magneta određenih stringom `s` duljine `n`. Drugi konstruktor prima jednu varijablu `V` tipa **`vector<string>`** u kojima se nalaze stringovi oblika `[+-]` ili `[-+]`, te stvara niz magneta iz vektora `V` navodeći ih od slijeva nadesno u istom poretku kao u vektoru `V`.
- *Operator* `[]`
  Prima indeks `i` (**`int`**) te vraća `i`-ti magnet po redu, gdje je nulti magnet onaj najlijeviji. Ako je `i<0` ili `i>=n` (gdje je `n` broj magneta u nizu), vratite prazninu `[ ]`.
  Ako je `i>=0` i `i<n`, omogućite promjenu magneta preko ovog operatora.
- *Operator* `()`
  Prima indeks `i` te vraća stanje (magnet ili prazninu) u legalnom nizu magneta koje se nalazi na `i`-tom mjestu po redu. Ako je `i<0` ili je `i` veći ili jednak od broja magneta i praznina u odgovarajućem legalnom nizu, vratite prazninu `[ ]`.
- *Operator* `|`
  Može djelovati između dvije varijable tipa `Magneti`, ili jedne varijable tipa `Magneti` i jednog stringa (koji je oblika `[+-]` ili `[-+]`), u bilo kojem redoslijedu. U svakom slučaju stvara novu varijablu tipa **`Magneti`** u kojoj su nanizani magneti iz prvog operanda, pa magneti iz drugog operanda.
- *Operator* `>`
  Djeluje između jedne varijable `M` tipa **`Magneti`** i jedne varijable `m` tipa **`int`**, redom. U varijabli `M` briše `m` najdesnijih magneta. Ako je `m` veći ili jednak od broja magneta u nizu, `M` postaje prazan niz magneta.
- *Operator* `<`
  Slično, kao i prošli operator, djeluje između jedne varijable `M` tipa **`Magneti`** i jedne varijable `m` tipa **`int`**, redom. U varijabli `M` briše `m` najlijevijih magneta. Ako je `m` veći ili jednak od broja magneta u nizu, `M` postaje prazan niz magneta.

- *Cast u* **`double`**
  Vraća omjer broja praznina i broja svih stringova (magneta i praznina) u legalnom nizu.
- *Ispis oblika* `cout << M`.
  Ispisuje legalan niz magneta, s razmacima između magneta. Jedini razmaci su oni u stringovima koji označavaju praznine. Nakon zadnjeg magneta ne smije biti zarez, ali mora biti ispis u novi redak. Oprez: budite jako pažljivi s ovom funkcijom i pridržavajte se svih njenih pravila, budući da već i jedan krivi znak prilikom ispisa može značiti da nećete dobiti bod za odgovarajući test primjer. Usporedite i s našim primjerom.

Napomene u vezi funkcija

- Spremite sučelje u datoteku **`magneti.h`**, a implementaciju u **`magneti.cpp`**.
- Omogućite ulančavanje svih operatora za koje je to moguće.

Opće napomene

- Struktura, funkcije i datoteke koje šaljete moraju se zvati *točno* onako kako je zadano u zadatku. Pazite na mala i velika slova!
- Trebate poslati samo sučelje i implementaciju. U datotekama koje šaljete *ne smije* se nalaziti funkcija main()!
- Nijedna funkcija *ne smije* ništa učitavati s tipkovnice ili neke datoteke, niti išta ispisivati na ekran ili u neku datoteku, osim ako nije drugačije rečeno.

Zadatak 15

Implementirajte klasu `Park` koja predstavlja tlocrt parka. `Park` sadrži neka stabla koja su predstavljena dvjema koordinatama, te visinom i radijusom krošnje (sva četiri podatka su pozitivni `int`-ovi). Neka su `stabla` jednostavna struktura koja sadrži samo ta 4 podatka i konstruktor. Klasa treba imati konstruktor koji prima `M, N`, dimenzije parka. Treba imati implementirane i sljedeće operatore, koji trebaju podržavati ulančavanje gdje god je to moguće:

- ispis na `cout` operatorom `<<` . Park se ispisuje kao "$(x_1,y_1,h_1),(x_2,y_2,h_2),\ldots,(x_n,y_n,h_n)$" (bez navodnika), sortirano silazno po visinama h_i. Možete pretpostaviti da neće biti dvoznačnosti (dva stabla iste visine). **Operator ne smije ispisivati razmake i prelaske u novi red! Nepridržavanje uputa za ispis može vam donijeti 0 bodova!**
- binarni operator |, koji svako stablo `(x,y,h)` iz desnog operanda presađuje u lijevi na iste koordinate (briše iz desnog, dodaje u lijevi) ako je to moguće (ako `0<=x<=M i 0<=y<=N`, i na tim koordinatama već ne postoji stablo). Neka povratni tip bude takav da omogući ulančavanje prvog operanda. Preopteretite | tako da ima i funkcionalnost `Park|Stablo i Stablo|Park`.
- binarni operator & koji vraća novi park, dimenzija `min(M1,M2), min(N1,N2)` koji na koordinate `(x,y)` postavi stablo visine `h` i krošnje `r` ako i samo ako oba parka imaju po stablo na tim koordinatama, i tad je `h=min(h1,h2), r=min(r1,r2)`. Ne mijenja operande.
- cast u `int`, koji vraća ukupnu visinu svih stabala u tom `Park`-u.
- cast u `bool`, koji vraća istinu ako postoje dva stabla čije se krošnje sijeku. Napomena: Krošnja je kugla sa središtem na visini `h-r`.
- unarni `--` koji briše jedno stablo, `--Park` izbacuje najviše stablo, `Park--` izbacuje stablo sa krošnjom najvećeg radijusa (ako ima više krošnji maksimalnog radijusa, izbacuje najviše među tim stablima). Ako je park prazan, onda ne treba ništa mijenjati.
- binarni > koji vraća istinu ako lijevi operand ima veću površinu od desnog, inače vraća laž.
- unarni - koji stvara novi park centralno simetričan operandu (s obzirom na središte parka, točku `(M/2,N/2)`).
- `Park[int]`, koji prima `int h` i vraća broj stabala te ili veće visine koja se nalaze u operandu. Ne mijenjati operand.
- operator `(int x, int y)`, koji vraća visinu stabla u parku na tim koordinatama. Ako su koordinate izvan dimenzija parka vraća -1, a ako nema stabla, vraća 0.

Operatori koji ne mijenjaju operand(e) trebaju se moći pozivati i na const referencama na Parkove.

Sučelje klase `Park` spremite u datoteku `park.h`, a implementaciju u datoteku `park.cpp`.

Primjer klijentskog programa

```
#include <iostream>
#include "plan.h"

using namespace std;

int main()
{
      Park stariPark(8,6), noviPark(6,5);
    stariPark|Stablo(1,1,6,2)|Stablo(1,2,5,2)|Stablo(2,4,7,1)|Stablo(4,5,3,1)|
Stablo(7,5,8,1);
    (noviPark|stariPark)|Stablo(5,2,4,2)|Stablo(7,6,1,1)|Stablo(5,2,3,4);
                              \\...zadnja dva dodavana stabla nisu dodana jer prvo ne
odgovara po
                              \\   koordinatama, a na mjestu (5,2) vec postoji stablo

      cout<< noviPark << endl;   \\(2,4,7),(1,1,6),(1,2,5),(5,2,4),(4,5,3)
      cout<< stariPark << endl;  \\(7,5,8)

      stariPark|Stablo(1,1,7,3)|Stablo(1,2,3,1);
      cout<<(noviPark & stariPark)<<endl;   \\(1,1,6),(1,2,3)

      int n=noviPark; cout<<n<<endl;   \\25  ...=7+6+5+4+3

      if(noviPark) cout<<"razraslo se"<<endl;   \\"razraslo se", jer npr Stablo(1,1,6,2)
i Stablo(1,2,5,2)
                                                \\ imaju međusobno presijecajuce krosnje.

      cout<<noviPark[4]<<" "<<noviPark[6]<<endl;   \\4 2

      --(noviPark--);
      cout<< noviPark<< endl;       \\(1,2,5),(5,2,4),(4,5,3)     ...jer izbaci redom
(1,1,6),(2,4,7),

      if(stariPark>noviPark)cout <<"lijevi ima vecu povrsinu"<<endl;

      cout<<(-noviPark)<<endl;  \\(5,3,5),(1,3,4),(2,0,3)



      cout<<noviPark(1,2)<<" "<<noviPark(1,1)<<endl;   \\5 0

      return 0;
}
```

Opće napomene

- Struktura, funkcije i datoteke moraju se zvati točno onako kako je zadano u zadatku. Pazite na mala i velika slova.
- Trebate poslati samo sučelje i implementaciju. U datoteci koju šaljete ne smije se nalaziti funkcija main()!
- Nijedna funkcija ne smije ništa učitavati s tipkovnice ili neke datoteke, niti išta ispisivati na ekran ili neku datoteku.

Zadatak 16

Implementirajte klasu `Plan` koja predstavlja turistov plan razgledavanja grada. Grad je predstavljen koordinatnim sustavom sa cjelobrojnim koordinatama. Klasa treba imati konstruktor koji prima cijele brojeve x_0, y_0, x_1, y_1 i stvara plan koji kreće iz (x_0, y_0) i završava u (x_1, y_1). Defaultne vrijednosti neka su dane samo za polazište, $x_0=0$, $y_0=0$. Treba imati implementirane i sljedeće operatore, koji trebaju podržavati ulančavanje gdje god je to moguće:

- ispis na `cout` operatorom <<. Plan se ispisuje kao "$(x_0, y_0) \rightarrow (x_1, y_1) \rightarrow \ldots \rightarrow (x_n, y_n)$" (bez navodnika), gdje turist polazi iz (x_0, y_0), i redom obilazi znamenitosti na koordinatama (x_1, y_1), ..., (x_n, y_n). Svaki plan razgledavanja će sadržavati barem dvije stanice. **Operator ne smije ispisivati razmake i prelaske u novi red! Nepridržavanje uputa za ispis može vam donijeti 0 bodova!**
- binarni operator +, koji od dva plana stvara jedan plan tako da na prvi plan razgledanja nastavlja drugi. Ako je zadnje odredište prvog plana jednako polazištu drugog, to odredište se ne smije pojaviti dvaput zaredom u novom planu, već samo jednom. Također treba implementirati funkcionalnosti: `pair<int,int>+Plan` **zamjenjuje** polazište (x_0, y_0) na početku plana, dok `Plan+pair<int,int>` **dodaje** odredište (x_{n+1}, y_{n+1}) na kraj. Operator treba promijeniti operand koji je tipa `Plan` te, u zadnjem slučaju zanemariti `pair<int,int>` ako su mu `(first, second)` jednaki zadnjem odredištu.
- cast u `int`, koji vraća duljinu plana razgledavanja zaokruženu na najveće cijelo.
- cast u `bool`, koji vraća istinu ako su sva odredišta u planu razgledavanja međusobno različita (odnosno, ako se nijedan (x_i, y_i) ne ponavlja), inače vraća laž.
- unarni `--` koji zamjenjuje poredak točno dva **uzastopna** odredišta, ona na `i`. i `(i+1)`. mjestu, ako time smanjuje duljinu plana razgledavanja. Pri tome `--Plan` uzima prvi takav `i>0` za koji je poboljšanje moguće, a `Plan--` posljednji takav `i`. Ako takvo poboljšanje nije moguće, onda ne treba ništa mijenjati.
- binarni < koji uspoređuje broj odredišta, tj. vraća istinu ako lijevi operand ima manji broj oredišta nego desni operand, inače vraća laž.
- unarni ~, koji vraća plan u kojem su izbačena višestruka ponavljanja istog odredišta. Početni plan mora ostati nepromijenjen.
- `Plan[int]`, koji vraća `pair` koordinata odredišta koje odgovara indeksu ([0] je polazište, [1] prva znamenitost itd...). Ako je indeks negativan, treba vratiti odgovarajuće odredište gledano od kraja ([-1] je konačno odredište, [-2] predzadnje, itd…). Smijete pretpostaviti da će traženo odredište uvijek postojati. Treba omogućiti i mijenjanje odredišta upotrebom operatora [] s lijeve strane pridruživanja.
- operator `()`, koji prima dva integera i vraća prvi redni broj znamenitosti ako se taj par koordinata nalazi u planu razgledanja, a inače -1.

Operatori koji ne mijenjaju operand(e) trebaju se moći pozivati i na const referencama na `Plan`ove.

Sučelje klase `Plan` spremite u datoteku `plan.h`, a implementaciju u datoteku `plan.cpp`.

Primjer klijentskog programa

```
#include <iostream>
#include "plan.h"
#include <utility>
using namespace std;

int main()
{
	Plan Ante(2,3,6,7), Mate(2,3);
	cout<< Mate << endl;	\\(0,0)->(2,3)
	cout<< Ante << endl;	\\(2,3)->(6,7)


	cout<< Mate +Ante<<endl;	\\(0,0)->(2,3)->(6,7)

	(make_pair(6,8)+(Mate+ make_pair(6,5)))+make_pair(6,5);
	cout<<Mate<<endl;		\\(6,8)->(2,3)->(6,5)	...promjena polazista, ne
dupla zadnju stanicu "->(6,5)->(6,5)"

	int n=Mate+Ante;
	cout<<n<<endl;	\\	5 \\=3+2

	cout<<(bool)Mate<<" "<<(bool)(Mate+Mate)<<endl;	\\true false

	(Ante+make_pair(3,3))+	make_pair(6,5)+make_pair(4,4)+make_pair(5,5);	\\Ante=(2,3)-
>(6,7)->(3,3)->(6,5)->(4,4)->(5,5)
	--Ante;
	cout<<Ante<<endl;	\\ (2,3)->(3,3)->(6,7)->(6,5)->(4,4)->(5,5)
				\\ isprobavanje pocinje tek od indeksa i=1, ne 0;
				\\ se smanjila za SQRT(32)+SQRT(13)-1-2
	Ante--;
	cout<<Ante<<endl;	\\(2,3)->(3,3)->(6,7)->(6,5)->(5,5)->(4,4)
				\\ duljina se smanjila za SQRT(5)-1

	cout<<(Mate<Ante)<<endl;	\\false
					\\Mate ima vise odredista
	Mate+make_pair(3,3); Mate=Mate+Mate; make_pair(3,3)+ Mate;
	cout<<Mate<<" "<<(~Mate)<<endl;	\\ (3,3)->(2,3)->(6,5)->(3,3)->(6,8)->(2,3)-
>(6,5)->(3,3)  (3,3)->(2,3)->(6,5)->(6,8)

	cout<<Mate[0].first<<" "<<Ante[-1].second<<endl;	\\3 4
							\\ 3 iz Matinog polazista (3,3) i 4
iz Antinog zadnjeg odredišta (4,4)

	cout<<Mate(2,3)<<" "<<Mate(6,5)<<" "<<Mate(2,4)<<endl;	\\1 2 -1
	return 0;
}
```

Opće napomene

- Struktura, funkcije i datoteke moraju se zvati točno onako kako je zadano u zadatku. Pazite na mala i velika slova.
- Trebate poslati samo sučelje i implementaciju. U datoteci koju šaljete ne smije se nalaziti funkcija main()!
- Nijedna funkcija ne smije ništa učitavati s tipkovnice ili neke datoteke, niti išta ispisivati na ekran ili neku datoteku.

Zadatak 17

Implementirajte klasu **`Reljef`** koja predstavlja tlocrt područja u obliku kvadrata na kojima se nalaze planine ili udoline kojima su visine/dubine cijeli brojevi. Ako je `n` duljina stranice tog kvadrata (u daljnjem tekstu: dimenzija reljefa), pozicije na tom tlocrtu označavamo s parom indeksa `(i,j)`, gdje su `0<=i,j<n`.

Popis i opis funkcija:

- *Konstruktori*
  Napravite (barem) dva konstruktora. Prvi konstruktor prima jednu `n` varijablu tipa **`int`** te stvara reljef dimenzije `n` u kojima su visine svih točaka nula. Drugi konstruktor prima dvije varijable `V1` i `V2` tipa **`vector<int>`** jednake duljine (`n`), te stvara reljef dimenzije `n` takav da se na poziciji `(i,j)` nalazi planina/udolina razine `V1[i]*V2[j]`.
- *Operator* `()`
  Prima dva indeksa `i` i `j`, te vraća razinu planine/udoline na mjestu `(i,j)` u tom reljefu ako se ta pozicija nalazi na reljefu, a inače `0`. Ako se to mjesto nalazi u reljefu, omogućite promjenu vrijednosti razine planine/udoline preko ovog operatora.
- *Operator* `^`
  Djeluje između varijabili `R` tipa **Reljef** te `m` tipa **`int`**, redom. U reljefu **`R`** mijenja razinu svih planina/udolina u stupcu s indeksom `r` za `k`, gdje su `r` i `k` (jedinstveni) ostatak i rezultat pri cijelobrojnom djeljenju broja `m` s dimenzijom `n`: `m=k*n+r`. Oprez: `m` (pa onda i `k`) može biti negativan, dok je `r` uvijek iz skupa `{0,1,..,n-1}`.
- *Operator* `>`
  Slično kao gornji operator, samo za retke. Djeluje između varijabili `R` tipa **`Reljef`** te `m` tipa **`int`**, redom. U reljefu `R` mijenja razinu svih planina/udolina u retku s indeksom `r` za `k`, gdje su `r` i `k` (jedinstveni) ostatak i rezultat pri cijelobrojnom djeljenju broja `m` s dimenzijom `n`: `m=k*n+r`. Oprez: `m` (pa onda i `k`) može biti negativan, dok je `r` uvijek iz skupa `{0,1,..,n-1}`.
- *Operator* `~`
  Unarni operator na varijabli tipa **`Reljef`**. U danom reljefu nalazi sve planine/udoline najveće razine te ih uvećava za 1.
- *Operator* `%`
  Djeluje između varijabli `R` tipa **`Reljef`** te `m` tipa **`int`**, redom. Vraća novu varijablu tipa **`Reljef`** dimenzije `m`, sastavljenu od presjeka zadnjih `m` stupaca i zadnjih `m` redaka reljefa `R`. Ako je `m` veći ili od dimenzije reljefa `R`, ostatak se reljefa (prvih nekoliko stupaca i redaka) popunjava planinama/udolinama razine 0.
- *Operator* `+`
  Djeluje između dviju varijabli `R1` i `R2` tipa **`Reljef`**. Vraća novu varijablu tipa **`Reljef`** dimenzije jednake maksimumu dimenzija reljefa `R1` i `R2`, koji na poziciji `(i,j)` ima planinu/udolinu razine `R1(i,j)+R2(i,j)`.
- *Cast u* **`double`**
  Vraća prosječnu razinu planina/udolina u reljefu.
- *Ispis oblika* `cout << R`
  Ispisuje tablicu reljefa, počevši s retkom razina planina/udolina na mjestima `(0,j)` redom, odvojene razmacima, a u zadnjem retkom razninama na mjestima `(n-1,j)`. Nakon zadnjeg

retka napravite ispis u novi redak. Nakon zadnjeg elementa u svakom retku nema razmaka. Oprez: budite jako pažljivi s ovom funkcijom i pridržavajte se svih njenih pravila, budući da već i jedan krivi znak prilikom ispisa može značiti da nećete dobiti bod za odgovarajući test primjer. Usporedite i s našim primjerom.

Napomene u vezi funkcija

- Spremite sučelje u datoteku **`reljef.h`**, a implementaciju u **`reljef.cpp`**.
- Omogućite ulančavanje svih operatora za koje je to moguće.

Opće napomene

- Struktura, funkcije i datoteke koje šaljete moraju se zvati *točno* onako kako je zadano u zadatku. Pazite na mala i velika slova!
- Trebate poslati samo sučelje i implementaciju. U datotekama koje šaljete *ne smije* se nalaziti funkcija main()!
- Nijedna funkcija *ne smije* ništa učitavati s tipkovnice ili neke datoteke, niti išta ispisivati na ekran ili u neku datoteku, osim ako nije drugačije rečeno.

Zadatak 18

`Hrana` je apstraktna klasa, iz koje su izvedene `Voce` i `Povrce`. Svaki objekt tipa `Hrana` pamti sljedeće podatke: `kolicina` (`int`, broj komada proizvoda), `ime` (`string`, ime proizvoda) i `prosjecnaTezina` (`double`, tezina u gramima nekog proizvoda; razlikuje se za objekte tipa `Voce` i `Povrce`). **Sve članske varijable su `private`.** Smijete pretpostaviti da će sve numeričke vrijednosti gore navedenih podataka biti nenegativne. Pretpostavljamo da svaki objekt tipa `Voce` teži prosječno 300 grama, dok svaki objekt tipa `Povrce` teži 500 grama.

Za objekte tipa `Voce` i `Povrce` implementirajte konstruktore koji primaju vrijednosti za varijable `kolicina` i `ime`, redom. Za sve tri varijable implementirajte desktruktor koji ispisuje poruku `"Unisten <tip>."`, gdje je `"<tip>"` jednak `Hrana`, `Voce` ili `Povrce`.

Implementirajte funkcije `postajeBiolosko` i `svePostajeBiolosko`. Prva funkcija je članska te pozvana na objektu tipa `Voce` mijenja `ime` proizvoda tako da mu na početak dodaje `"bio "`, dok na objektu tipa `Povrce` mijenja ime proizvoda tako da mu na kraj dodaje `" organskog podrijetla"`. Druga funkcija je static te mijenja ime svim varijablama koje postoje tijekom izvođenja programa na gore opisan način.

Implementirajte globalnu funkciju `MasaKosarice` koja može primiti varijablu `kosarica` tipa `stack<Hrana*>`. Vraća ukupnu masu svih proizvoda (dobivena kao umnožak prosječne težine i količine proizvoda) koji se nalaze u `kosarici`. Funkcija mora biti globalna, ne smije biti static niti friend.

Napišite i glavni program u kojem ćete stvoriti barem dva objekta tipa `Voce`, i barem dva objekta tipa `Povrce`, te testirati sve navedene funkcije.

Sakrijte "od javnosti" sve podatke koje možete i izbjegavajte dupliciranje dijelova koda - u protivnom možete izgubiti bodove! Zabranjeno je koristiti globalne varijable! Potrebno je pobrinuti se da se `const` koristi svugdje gdje je moguće te da se reference koriste kod prijenosa klasa kao parametara u funkcije gdje je moguće. Smijete implementirati i dodatne članske funkcije i varijable, no i dalje sve članske varijable trebaju biti private. Osigurajte da oslobodite svu dinamički alociranu memoriju.

Zadatak 19

Implementirajte klasu `funkc` čiji objekti predstavljaju funkcije čija domena je skup $\{0, 1, 2, \ldots, n\}$, a kodomena skup cijelih brojeva.
Klasa treba imati sljedeće elemente:

- `int n` – broj n iz gornjeg teksta koji određuje domenu funkcije,
- `int* maps` – niz cijelih brojeva takav da ako je `maps[i] == k`, tada funkcija preslikava broj i u broj k, odnosno vrijedi $f(i) = k$.

Klasa treba imati sljedeće funkcije:

- `funkc()` – defaultni konstruktor koji kreira funkciju s domenom $\{0\}$ za koju je $f(0) = 0$,
- `funkc(int m, int a0)` – konstruktor koji kreira konstantnu funkciju s domenom $\{0, 1, \ldots, m\}$ za koju je $f(x) = a0$ za svaki x iz domene,
- `funkc(vector<int> V)` – konstruktor koji kreira funkciju koja broju i pridružuje broj `V[i]` za sve $0 \leq i <$ `V.size()`, dakle, $f(i) =$`V[i]`. Na ostalim brojevima funkcija nije definirana,
- `funkc(const funkc &a)` – copy konstruktor koji kreira funkciju identičnu funkciji `a`, pazeći pritom da se ne dogodi kopiranje adrese `maps`, alocirajte odvojenu memoriju,
- `~funkc()` – destruktor koji oslobađa zauzetu memoriju `maps`.

Omogućite rad sljedećih operatora:

- `operator+` – zbrajanje funkcija f i g, vraća funkciju $(f + g)$ ako f i g imaju iste domene (tada i zbroj ima tu istu domenu), inače ispisati *Nedozvoljeno zbrajanje!* i vratiti funkciju nastalu pozivom defaultnog konstruktora,
- `operator*` – komponiranje funkcija f i g, vraća funkciju $f \circ g$ ako je slika funkcije g podskup domene funkcije f (dakle, ako je kompozicija dobro definirana). Inače ispisati *Nedozvoljena kompozicija!* i vratiti funkciju nastalu pozivom defaultnog konstruktora,
- `operator-` - operator unarnog minusa nad f, treba vratiti $-f$,
- `operator-` - operator oduzimanja, za f i g treba vratiti funkciju $(f - g)$ ako f i g imaju iste domene (tada i razlika ima tu istu domenu), inače ispisati *Nedozvoljeno oduzimanje!* i vratiti funkciju nastalu pozivom defaultnog konstruktora,

- `operator«` - operator ispisivanja funkcije f. Npr., ako je domena od f jednaka $\{0, 1, 2\}$ i vrijedi $f(0) = 3$, $f(1) = -2$ i $f(2) = 0$, tada treba ispisati: *f(0) = 3, f(1) = -2, f(2) = 0*, svaki string u svom redu. Bez obzira na naziv varijable, u ispisu uvijek možete koristiti f za funkciju,

- `operator=` - operator pridruživanja koji jednoj varijabli tipa `funkc` pridružuje vrijednost druge varijable tipa `funkc`. Pripazite da se ne dogodi kopiranje adrese `maps`, alocirajte odvojenu memoriju,

- `operator+=` - operator koji zadanu funkciju uvećava za funkciju g,

- `operator++` - operatori prefiksnog i postfiksnog inkrementa koji zadanoj funkciji povećavaju vrijednost u svakom elementu domene za 1.

- `operator()` – operator koji prima jedan cijeli broj x koji vraća vrijednost funkcije u broju x i tako omogućava npr. korištenje naredbe `f(5);`, ako x nije u domeni funkcije, vratiti 0 i ispisati *Element nije u domeni!*.

Ne trebate paziti na složenost. Omogućite ulančavanje gdje god je moguće. Svi elementi klase mogu biti public. Napišite i neki kratki main koji testira barem 2 gore navedena operatora.

Zadatak 20

`Hrana` je apstraktna klasa, iz koje su izvedene `Meso` i `Mlijeko`. Svaki objekt tipa `Hrana` pamti sljedeće podatke: `kolicina` (`double`, opisuje količinu/broj komada proizvoda), `rokTrajanja` (`int`, koliko još dana vrijedi proizvod) i `cijena` (`int`, cijena po komadu, razlikuje se za objekte tipa `Meso` i `Mlijeko`). **Sve članske varijable su `private`.** Smijete pretpostaviti da će sve vrijednosti gore navedenih podataka biti nenegativne. Svaki komad mesa košta 50kn, dok mlijeko košta 10kn.

Za objekte tipa `Meso` i `Mlijeko` implementirajte konstruktore koji primaju vrijednosti za varijable `kolicina` i `rokTrajanja`, redom. Za sve tri varijable implementirajte desktruktor koji ispisuje poruku `"Unisten <tip>."`, gdje je `"<tip>"` jednak `Hrana`, `Meso` ili `Mlijeko`.

Implementirajte funkcije `produljiRok` i `produljiRokSvima`. Prva funkcija je članska te pozvana na objektu tipa `Meso` udvostručuje preostali rok trajanja, dok na objektu tipa `Mlijeko` produljuje rok trajanja za jedan dan. Druga funkcija je static te mijenja rok trajanja svim varijablama koje postoje tijekom izvođenja programa na gore opisan način.

Implementirajte globalnu funkciju `CijenaHladnjaka` koja može primiti varijablu `hladnjak` tipa `queue<Hrana*>`. Vraća ukupnu cijenu svih proizvoda (dobivena kao umnožak cijene i količine proizvoda) koji se nalaze u `hladnjaku`. Po završetku izvođenja programa, svaki proizvod u `hladnjaku` mora imati rok trajanja postavljen na 1 dan. Funkcija mora biti globalna, ne smije biti static niti friend.

Napišite i glavni program u kojem ćete stvoriti barem dva objekta tipa `Meso`, i barem dva objekta tipa `Mlijeko`, te testirati sve navedene funkcije.

Sakrijte "od javnosti" sve podatke koje možete i izbjegavajte dupliciranje dijelova koda - u protivnom možete izgubiti bodove! Zabranjeno je koristiti globalne varijable! Potrebno je pobrinuti se da se `const` koristi svugdje gdje je moguće te da se reference koriste kod prijenosa klasa kao parametara u funkcije gdje je moguće. Smijete implementirati i dodatne članske funkcije i varijable, no i dalje sve članske varijable trebaju biti private. Osigurajte da oslobodite svu dinamički alociranu memoriju.

Zadatak 21

Implementirajte klasu `poly` čiji objekti predstavljaju polinome s cjelobrojnim koeficijentima.
Klasa treba imati sljedeće elemente:

- `int deg` – broj koji pamti stupanj pripadnog polinoma,
- `int* coefs` – niz cijelih brojeva takav da ako je `coefs[i] == k`, tada je u pripadnom polinomu koeficijent uz x^i jednak k.

Klasa treba imati sljedeće funkcije:

- `poly()` – defaultni konstruktor koji kreira konstantan polinom za koji vrijedi $P(x) = 0$,
- `poly(int a0)` – konstruktor koji kreira konstantan polinom za koji vrijedi $P(x) = a0$,
- `poly(vector<int> V)` – konstruktor koji kreira polinom čiji je koeficijent uz x^i jednak `V[i]` za sve $0 \leq i <$ `V.size()`,
- `poly(const poly &a)` – copy konstruktor koji kreira polinom identičan polinomu a, pazeći pritom da se ne dogodi kopiranje adrese `coefs`, alocirajte odvojenu memoriju,
- `~poly()` – destruktor koji oslobađa zauzetu memoriju `coefs`.

Omogućite rad sljedećih operatora:

- `operator+` – zbrajanje polinoma $P(x)$ i $Q(x)$, treba vratiti polinom $P(x) + Q(x)$,
- `operator*` – množenje polinoma $P(x)$ i $Q(x)$, treba vratiti polinom $P(x) \cdot Q(x)$,
- `operator-` - operator unarnog minusa nad $P(x)$, treba vratiti polinom $-P(x)$,
- `operator-` - operator oduzimanja, za $P(x)$ i $Q(x)$ treba vratiti polinom $P(x) - Q(x)$,
- `operator«` - operator ispisivanja polinoma $P(x)$. Ispisujte koeficijente od nižeg stupnja prema višem te nemojte ispisati višak znakova. Npr. ako je $P(x) = -x^2 + 1$, naredba `cout«P;` treba ispisati $1 - x^2$, a ne $1 + 0x + -1x^2$,

- `operator=` - operator pridruživanja koji jednoj varijabli tipa `poly` pridružuje vrijednost druge varijable tipa `poly`. Pripazite da se ne dogodi kopiranje adrese `coefs`, alocirajte odvojenu memoriju,
- `operator+=` - operator koji zadani polinom uvećava za polinom $Q(x)$,
- `operatori++` - operatori prefiksnog i postfiksnog inkrementa koji zadanom polinomu povećavaju slobodni član za 1.

Ne trebate paziti na složenost. Omogućite ulančavanje gdje god je moguće. Svi elementi klase mogu biti public. Napišite i neki kratki main koji testira barem 2 gore navedena operatora.

Zadatak 22

Implementirajte klasu `Racunalo` koja predstavlja računalo u nekoj mreži. Svako računalo ima jedinstveno ime (jedinstvenost ne morate provjeravati). Računala se mogu spajati s drugim računalima te slati i primati poruke. Svako računalo može se spojiti s ograničenim brojem drugih računala. Također, svako računalo ima i zapisnik (niz stringova) u kojem bilježi spajanja i odspajanja te primanje i slanje poruka. Smatramo da je svako računalo spojeno samo sa sobom i da se ne može odspojiti.

Klasa treba imati sljedeće konstruktore i operatore:

- `Racunalo R(string ime, int brVeza)` - stvara novo računalo s imenom `ime` koje može biti spojeno s najviše `brVeza` (ne računajući samo sebe) računala istovremeno. Zapisnik je na početku prazan.
- `zapis >> R`: dodaje novi unos u zapisnik, `zapis` je tipa `string`
- `R--` : briše zadnji unos u zapisniku
- `R1 | R2` : spaja računalo `R1` s računalom `R2`. Pritom, ukoliko su `R1` ili `R2` dosegli maksimalni broj veza ili ako su već spojeni, ništa se ne događa. Ukoliko su uspješno spojeni, ime računala na koje se spajaju unosi se u zapisnik uz prefiks "+" (za oba računala).
- `R1 / R2` : odspaja računala `R1` i `R2`, za oba računala u zapisnik zapisuje ime računala od kojeg se odspaja uz prefiks "-". Ne radi ništa ako računala nisu spojena.
- `!R` : odspaja računalo `R` od svih računala s kojima je spojeno nakon čega mu briše cijeli zapisnik.
- `R[ime]`: vraća `true` ako je računalo `R` spojeno s računalom imena `ime`, `false` inače.
- `R1(poruka, ime)`: šalje poruku na računalo imena `ime`. Ukoliko računala nisu spojena ne radi ništa. Ako su spojena, u zapisnik `R2` se dodaje poruka oblika "ime_posiljatelja: poruka", a u zapisnik `R1` "poruka -> ime_primatelja". Vraća `true` ako su računala spojena, `false` inače.
- `cout << R` : ispisuje naziv računala `R` i njegov zapisnik, tako da je svaki zapis u svom redu.
- `R1 > R2` : vraća `true` ako računala na koja je spojen `R2` čine podskup računala na koja je spojen `R1`, `false` inače.
- `R1 == R2` : vraća `true` ako su `R1` i `R2` spojeni na ista računala, `false` inače.
- cast u `int` : vraća broj različitih računala do kojih dano računalo može pristupiti u najviše 2 koraka. Npr. ako je računalo `R0` spojeno s računalom `R1` a `R1` je spojeno s `R2`, tada računalo `R0` može pristupiti računalu `R1` u jednom a `R2` u dva koraka.

Zadatak 23

Znanstvenici jednog poznatog instituta istražuju bakterije koje proizvode energiju. Implementirajte hijerarhiju klasa `Bakterija`, `BakterijaCO`, `BakterijaHC` i `BakterijaHCUV` koje opisuju vrste bakterija koje su znanstvenici proizveli. Klase moraju zadovoljavati sljedeća svojstva:

- `Bakterija` je apstraktna klasa sa funkcijama `hrani` i `energija`.
- Svaka `Bakterija` ima funkciju `hrani`. Ona prima jedan znak koji predstavlja vrstu hrane kojom smo nahranili bakteriju. Bakterije pamte redom hranu kojom smo ih nahranili (npr. nakon 10 poziva funkcije `hrani` s odgovarajućim parametrima bi neka bakterija zapamtila niz `COCHPCOHHH`).
- Bakterije proizvode energiju na različite načine. `BakterijaCO` će proizvesti 1W energije za svaki niz `CO` u hrani koju je pojela (npr. za niz iz prethodne točke bi proizvela 2W jer imamo 2 niza `CO`). `BakterijaHC` će proizvesti 1W energije ako bilo gdje u pojedenoj hrani pronađe jedno slovo `H` i jedno slovo `C` (npr. za niz iz prethodne točke bi proizvela 3W jer imamo 4 slova `H` ali samo 3 slova `C`.). Funkcija `energija` vraća količinu proizvedene energije u dosad pojedenoj hrani. Nakon poziva ove funkcije, sva pojedena hrana nestaje iz bakterije.
- Bakterije tipa `BakterijaCO` mogu preotimati hranu drugim bakterijama iste vrste. U toj klasi postoji funkcija `preotmi` koja kao parametar prima referencu na drugu bakteriju tipa `BakterijaCO`. Ako su `X` i `Y` varijable tipa `BakterijaCO`, onda nakon poziva `X.preotmi(Y)` sva hrana koju je imala bakterija `Y` pripada bakteriji `X`, a bakterija `Y` više nema hrane. Niz hrane koju je imala `Y` dolazi se na kraj niza hrane kojeg je imala `X`.
- `BakterijaHCUV` je posebna podvrsta od `BakterijaHC`. Znanstvenici su otkrili da jedina razlika nastaje kada bakteriju tipa `BakterijaHCUV` obasjamo UV-svjetlom. Naime, sve bakterije tipa `BakterijaHCUV` koje su obasjane UV-svjetlom dijele hranu: ako jednu takvu bakteriju nahranimo pomoću funkcije `hrana`, automatski istu tu hranu dobivaju sve bakterije tog tipa koje su obasjane UV svjetlom. U klasi `BakterijaHCUV` postoji funkcija `uv` koja prima jedan string. Ako je taj string `"on"`, onda smo bakteriju na kojoj smo pozvali funkciju obasjali UV-svjetlom; ako je taj string `"off"`, onda smo maknuli UV-svjetlo. Na početku, niti jedna bakterija nije obasjana UV-svjetlom.
- sve bakterije imaju destruktor. On ispisuje hranu koju je bakterija pojela od zadnjeg poziva funkcije `energija`, te ukupnu količinu energije koju je bakterija proizvela svim pozivima funkcije `energija`. Za `BakterijaHCUV`, destruktor još ispisuje je li bakterija obasjana UV-svjetlom (string `"on"` ili `"off"`).

Znanstvenici imaju 10 `BakterijaCO`, 15 `BakterijaHC` i 20 `BakterijaHCUV`. Prvo su nahranili 3 bakterije tipa `BakterijaCO` (odaberite po volji hranu), te dali prvoj da preotme hranu drugoj i trećoj. Nad prvih 5 `BakterijaHCUV` su upalili UV-svjetlo. Zatim su sve bakterije su nahranili istom hranom (odaberite po volji), te pogledali koliko su energije proizvele.
U glavnom programu implemetirajte sve ove operacije, tako da sve bakterije za vrijeme poziva funkcija `hrana` i `energija` iz prethodne rečenice budu u zajedničkoj kolekciji (`vector`).

Sakrijte podatke koje korisnik funkcija članica klasa ne treba vidjeti. Izbjegnite nepotrebno dupliciranje koda.

Cjelokupni program spremite pod imenom `zadatak1.cpp`.

Zadatak 24

Dana je klasa `Recenica`, iz koje je izvedena klasa `Palindrom`, iz koje je izvedena klasa `Rij4c`. Svaki objekt tih klasa pamti podatak `s` (protected!) tipa `string`, koji predstavlja rečenicu. U njoj se pojavljuju mala slova engleske abecede i razmaci, s izuzetkom na prvom mjestu (gdje dolazi veliko slovo engleske abecede), te na zadnjem mjestu (gdje dolazi točka). Možete pretpostaviti da će sve riječi u rečenici biti duljine barem 2, a sve rečenice sadržavati barem 4 slova. Objekt tipa `Palindrom` pamti rečenice koje se jednako čitaju slijeva na desno i zdesna na lijevo kada im se maknu svi razmaci i točka. Objekt tipa `Rij4c` pamti palindromne rečenice sastavljene od jedne riječi od 4 slova (i točke na kraju).

Za sva tri objekta implementirajte konstruktor koji prima jedan `string`. Smijete pretpostaviti da će sadržavati samo slova engleske abecede i razmake. Pozvan na tipu `Recenica`, sprema taj `string` u `s` tako da mu nadoda točku na kraj, te po potrebi promijeni velika i mala slova u rečenici (veliko slovo dolazi isključivo na prvom mjestu). Konstruktor za objekt tipa `Palindrom` dodatno promijeni zadnjih $\lfloor n/2 \rfloor$ slova rečenice (gdje je n broj slova u rečenici) kako bi rečenica postala palindrom. Konstruktor za objekt tipa `Rij4c` dodatno po potrebi skraćuje `s` tako da u obzir uzme isključivo prva dva i zadnja dva slova dobivene palindromne rečenice.

Primjer: u slučaju da u konstruktoru ulazni `string` ima vrijednost `"Ana voli Makarsku"`, konstruktor za `Recenicu` pretvorio bi ju u `"Ana voli makarsku."`, konstruktor za `Palindrom` pretvorio bi ju u `"Ana voli milovana."`, dok bi konstruktor za `Rij4c` stvorio `"Anna."`

Implementirajte funkciju `ispis` koja na ekran ispisuje danu rečenicu iz `s`. Za sva tri tipa implementirajte destruktor koji ispisuje poruku `"Unisten <tip>."`, gdje je `"<tip>"` jednak `Recenica`, `Palindrom` ili `Rij4c`.

Implementirajte funkciju `popisRij4ci` (člansku za `Rij4c`) koja ispisuje sve objekte tipa `Rij4c` koje postoje tijekom izvođenja programa.

Implementirajte virtualnu funkciju `promijeni` koja u `s` mijenja prvo (u slučaju poziva na objektu tipa `Palindrom`), odnosno prva dva (u slučaju tipa `Rij4c`) slova u njihove sljedbenike engleske abecede. Možete pretpostaviti da se funkcija neće pozivati na rečenicama koje sadržavaju slovo `'z'` (i `'Z'`). Pobrinite se da promijenite i ostala odgovarajuća slova u `s` da bi se očuvalo svojstvo palindroma.

Napišite i glavni program koji stvara barem jedan objekt svakog od tipova i testira sve funkcije.

Sakrijte "od javnosti" sve podatke koje možete i izbjegavajte dupliciranje dijelova koda. Zabranjeno je koristiti globalne varijable! Potrebno je pobrinuti se da se `const` koristi svugdje gdje je moguće te da se reference koriste kod prijenosa klasa kao parametara u funkcije gdje je moguće. Smijete implementirati i dodatne članske funkcije i varijable. Oslobodite svu dinamički alociranu memoriju.

Zadatak 25

Implementirajte klasu `Tenk` koja predstavlja tenk u borbi. Tenkovi se nalaze na cjelobrojnim koordinatama ravnine, te mogu pucati, a onda i uništavati jedni druge. Svaki objekt tipa `Tenk` pamti jedinstveno `ime` (`string`), koordinate (`x`,`y`) (`int`), i `int` varijable `municija` (broj topovskih kugli), `domet` (do koje udaljenosti tenk može pucati) i `oklop` (razina oštećenosti tenka; ako je `oklop<=0`, tenk je uništen). Dodatno, objekti tipa `Tenk` pamte i popis imena drugih tenkova koje su oštetili pucanjem.

Klasa treba imati sljedeće članice:

- Konstruktor koji prima podatke `ime`, `x`, `y` i `municija`, redom. Za svaki novostvoreni tenk, varijable `domet` i `oklop` postavljaju se na `3`.
- Operatori inkrementa(`++`) i dekrementa(`--`): povećavaju/smanjuju koordinate za 1, pri čemu prefiksni operatori pomiču tenk duž x osi, a postfiksni duž y osi.
- Operator `!` koji stvara kopiju tenka na kojem je pozvan ovaj operator. Varijabla `ime` dobiva sufiks `" kopija"`; tenk je centralno simetričan u odnosu na ishodište u odnosu na tenk koji je pozvao ovaj operator; kopiraju mu se sadržaji varijabli `municija`, `domet` i `oklop`, dok mu je popis tenkova na koje je pucao prazan.
- Operator `[]`. U slučaju poziva oblika `T1[n]`, gdje je `T1` tipa `Tenk`, a `n` tipa `int`, uvećava razinu municije tom tenku `n` puta.
- Operator `()`. U slučaju poziva oblika `T1(n)`, gdje je `T1` tipa `Tenk`, a `n` tipa `int`, uvećava domet tom tenku za `n`, pritom umanjujući municiju za `n`. Ako je `n>municija`, funkcija ne radi ništa.
- Operator `>`. U slučaju poziva oblika `T1>T2`, gdje su `T1` i `T2` tipa `Tenk`, `T1` pokušava pucati na `T2`, ako `T1` ima municije i nije uništen. U tom slučaju municija mu se smanjuje za 1. Ako se `T2` nalazi u dometu od `T1` (udaljenost im je manja ili jednaka `T1.domet`), tenk `T2` biva pogođenim, čime mu se varijabla `oklop` smanjuje za jedan.
- Operator *castanja* u `bool` koji vraća vraća `true` ako tenk nije uništen.
- Omogućite ispis na ekran pomoću `"cout <<"` tako da ispišete ime tenka, a zatim i imena svih tenkova na koje je taj tenk pucao (bez ponavljanja).

Omogućite ulančavanje gdje god je moguće. Svi elementi klase mogu biti public. Napišite i neki main koji testira barem 2 gore navedena operatora.

Zadatak 26

Implementirajte klasu `Survivor` koja predstavlja izletnika/kampera čiji se izlet zakomplicirao i sad treba preživljavati. `Survivor` sadrži podatke o kapacitetu ruksaka (masa u gramima, prirodan broj) i sadržaju ruksaka (popis predmeta u ruksaku, `Predmet` je jednostavna struktura koja ima samo podatke `naziv (string), masa (int), korisnost (int)` te konstruktor. `Survivor` sadrži i `bool` podatak `suradnja`. Klasa treba imati konstruktor koji prima `kapacitet` i `suradnja` (default=`false`) i stvara `Survivora` sa tim podacima koji u ruksaku već ima 3 šibice ("sibica", mase 1, korisnosti 100). Treba imati implementirane i sljedeće operatore, koji trebaju podržavati ulančavanje gdje god je to moguće:

- ispis na `cout` operatorom `<<` . Survivor se ispisuje kao "$predmet_1$,$masa_1$,$korisnost_1$;$predmet_2$,$masa_2$,$korisnost_2$;...;$predmet_n$,$masa_n$,$korisnost_n$;" (bez navodnika), sortirano po imenima predmeta. Predmeti istog imena će imati istu masu i korisnost. **Operator ne smije ispisivati razmake i prelaske u novi red! Nepridržavanje uputa za ispis može vam donijeti 0 bodova!**
- binarni operator +, koji simulira susret dva Survivora na slijedeći način: ako oba imaju `suradnja==true`, obojici se u ruksak doda po ("hrana", 500, 300), ako može stati. Ako su im modovi suradnje različiti, onaj kojemu je status `suradnja==false` , ukrade onom sa statusom `suradnja==true` najkorisniji predmet iz ruksaka koji može stati u njegov ruksak (možete pretpostaviti da neće biti dvoznačnosti). Ako je obojici `suradnja==false`, obojica izgube po najkorisniji predmet iz svojih ruksaka. Neka povratni tip bude takav da omogući ulančavanje prvog operanda.
- Omogućite i funkcionalnosti `Predmet+Survivor` i `Survivor+Predmet` - dodavanje jednog `predmeta`, omogućite ulančavanje sa operandom koji je `Survivor`
- unarni operator ~ koji mijenja mod suradnje operanda, omogućiti ulančavanje.
- cast u `int`, koji vraća ukupnu korisnost predmeta u ruksaku.
- cast u `bool`, koji vraća istinu ako je ukupna masa predmeta veća ili jednaka pola kapaciteta ruksaka, inače vraća laž.
- unarni `--` koji izbacuje iz popisa jedan predmet, --Survivor izbacuje najteži predmet, Survivor-- izbacuje najmanje koristan predmet. Ako je ruksak prazan, onda ne treba ništa mijenjati. Za oba omogućite ulančavanje.
- binarni > koji vraća istinu ako lijevi operand ima neki predmet kojeg desni operand nema, inače vraća laž.
- `Survivor[int i]`, koji vraća naziv `i.` predmeta po težini ([0] je najteži predmet, [1] drugi itd...). Ako je indeks negativan, treba vratiti odgovarajući predmet gledano od kraja ([-1] je zadnji, [-2] predzadnji, itd…). Smijete pretpostaviti da će traženi predmet uvijek postojati.
- operator `()`, koji prima `string` i vraća ukupan broj predmeta tog imena u ruksaku.

Operatori koji ne mijenjaju operand(e) trebaju se moći pozivati i na const referencama na Survivore.

Sučelje klase `Survivor` spremite u datoteku `survivor.h`, a implementaciju u datoteku `survivor.cpp`.

Primjer klijentskog programa

```
#include <iostream>
#include "suvivor.h"

using namespace std;

int main()
{
      Survivor Ante(20000,true), Mate(10000);
```

```
	cout<< Mate << endl;	\\sibica,1,100;sibica,1,100;sibica,1,100;

	Ante+Mate;
	cout<<Ante<<endl<<Mate<<endl; \\sibica,1,100;sibica,1,100;
	                              \\sibica,1,100;sibica,1,100;sibica,1,100;sibica,1,100;
	Ante+(~Mate);
	cout<<Ante<<endl<<Mate<<endl; \\hrana,500,300;sibica,1,100;sibica,1,100;
	                              \\
hrana,500,300;sibica,1,100;sibica,1,100;sibica,1,100;sibica,1,100;

	(~Ante)+(~Mate);
	cout<<Ante<<endl<<Mate<<endl; \\sibica,1,100;sibica,1,100;
	                              \\sibica,1,100;sibica,1,100;sibica,1,100;sibica,1,100;

	int n=Ante;cout<<n<<endl; \\200

	if(Mate)cout<<"preopterecen"<<endl; else cout<<"nije preopterecen"<<endl;	\\nije
preopterecen
	Ante+Predmet("rostilj", 8000, 500);
	if(Mate)cout<<"preopterecen"<<endl; else cout<<"nije preopterecen"<<endl;	\\
preopterecen

	cout<<(Mate>Ante)<<endl;	\\1
	cout<<Mate[0]<<endl;	\\rostilj

	(--Mate)--;
	cout<<Mate<<endl; \\sibica,1,100;sibica,1,100;sibica,1,100;
	                  \\izbacio je "rostilj" i "sibicu"

	cout<<(Mate>Ante)<<endl;	\\0

	cout<<Mate("sibica")<<" "<<Mate("rostilj")<<endl;	\\3 0

	return 0;
}
```

Opće napomene

- Struktura, funkcije i datoteke moraju se zvati točno onako kako je zadano u zadatku. Pazite na mala i velika slova.
- Trebate poslati samo sučelje i implementaciju. U datoteci koju šaljete ne smije se nalaziti funkcija main()! Nijedna funkcija ne smije ništa učitavati s tipkovnice ili neke datoteke, niti išta ispisivati na ekran ili neku datoteku.